

Secret and Credential Isolation

For devs, security, and operators · ~12 min read

Why secrets in AI workflows need special handling

A prompt is not a private message. The moment you type a secret into an agent's input box, paste it into a chat window, or let a coding assistant read a file that contains it, that secret has left your trust boundary and entered somebody else's. It lands in the model provider's logs. It lands in the agent framework's telemetry. It lands in the local IDE plugin's "recent context" cache. It lands in the chat transcript your teammate can search tomorrow. A long-lived admin token that takes this journey should be treated, for every purpose that matters, as public.

Every prompt is a potential egress. Secrets in context = secrets in logs.

This is not a hypothetical. Transcripts of AI sessions routinely end up in bug reports, shared gists, customer support tickets, and training data opt-ins that nobody read carefully. The blast radius of a leaked chat transcript is whatever the most powerful credential in that transcript was. If you never put an admin token in there, the worst-case transcript is boring. If you did, the worst case is an incident with your name on it.

The rest of this lesson is a small, opinionated discipline for never putting the dangerous ones in there in the first place.

Scope is the primary control

Least privilege, applied to credentials, is called scoping. A scoped token can read one repository, not the whole org. A scoped API key can call one endpoint family, not every API the provider offers. A scoped database role can `SELECT` from two tables in one schema, not drop the database. Scoping is the control that survives a leak. Rotation is the control that limits how long a leak is useful. Scope matters first, because an unscoped token that rotates every hour is still an admin token for an hour — and an

hour is a long time when "admin" means "exfiltrate every repo and open a backdoor in the infra account."

The question to ask for every credential in an agent's context is not "is this token safe?" but "what does the world look like if this token is printed on the front page of a newspaper tomorrow?" If the answer is "nothing much — it can read one repo's issues for the next fifteen minutes," you are fine. If the answer involves phone calls to your compliance team, the token was out of scope. Mint a narrower one.

Modern providers have mostly caught up. GitHub has fine-grained PATs, Stripe has restricted keys, OpenAI and Anthropic have per-project keys with spend caps, AWS has STS AssumeRole with session policies. There is almost never a technical reason in 2026 to give an agent an org-wide admin key. The only reason it still happens is friction — the scoped path has three more steps than the unscoped path, and whoever set up the agent took the short path. Your job is to notice that and take the longer path anyway.

Per-project `.env` hygiene

The unit of isolation for an interactive developer environment is the per-project `.env` file. One project, one file, the minimum set of credentials that project needs, each credential scoped to that project's needs. The file lives in the project directory, not in your home directory. It is `chmod 600` so nothing else on your machine can casually read it. It is listed in `.gitignore` before the first secret is written to it, and every project has a matching `.env.example` with placeholder values so new contributors know what to fill in without learning the real values.

What a per-project `.env` is not: a shared vault. Never copy an `.env` between projects. If project B needs the same API that project A uses, mint a second scoped credential specifically for project B. That way, when one project is compromised, the blast does not automatically extend to the other. It is tempting to share — it saves thirty seconds — and it is exactly how a single stolen `.env` turns into a week of incident response touching every project you own.

A second rule: never paste the contents of an `.env` into a chat window, a ticket, a Slack DM, or an AI prompt. If a tool or a coworker needs to know what variables are expected, share the `.env.example`. If they need real values, mint them their own scoped credentials through whatever the provider's normal path is. "Here is my `.env`" is always the wrong answer.

The never-to-an-agent list

Some credentials simply do not belong in agent context under any circumstances. If you find yourself about to put one of these into a prompt, into an MCP server config, or into an agent's environment, stop and mint something narrower instead.

- **Long-lived personal access tokens** — classic GitHub/GitLab PATs with repo-admin, org-admin, or workflow scopes. Use fine-grained tokens scoped to a single repository with only the permissions that repository's agent needs.
- **Root API keys** — the default organisation key for OpenAI, Anthropic, AWS, Stripe, SendGrid, or any other provider. Use per-project keys with spend caps and scoped permissions.
- **SSH private keys** — anything in `~/.ssh/`, especially `id_rsa` or `id_ed25519`. If an agent needs to push code, mint a deploy key scoped to one repo, or let it push over HTTPS with a fine-grained token.
- **Cloud credentials with console access** — IAM user access keys, GCP service account JSONs that map to human-equivalent roles, Azure service principal secrets with owner permissions. Use assumed-role sessions with short TTLs.
- **JWT signing secrets, session secrets, cookie encryption keys** — these belong to the service, not to any caller of the service. An agent never needs to see them.
- **Anything labelled "prod"** that you cannot explain one narrow reason for the agent to touch.

If a credential on this list shows up in an agent log, the correct response is to rotate it immediately, not to argue about whether it was technically exposed. The technical question is unanswerable from the outside; the operational question — "is this credential now untrusted?" — has an easy answer.

Rotation

Rotation cadence is a function of blast radius and usage pattern. Three tiers cover most practical cases.

Per-run, for agent runs. If an agent runs to completion and then exits, its credentials should be minted at the start of the run and revoked at the end. TTL in minutes. The credential exists for the lifetime of one job, and when the job ends it is worthless to

anyone who finds it in a log. This is the gold standard for autonomous agent tokens, and it is also the easiest rotation policy to automate because the rotation points are exactly where the lifecycle events already are.

Daily, for engagement-bound tokens. If a token is attached to an ongoing engagement — a customer project, a sprint, an investigation — mint it fresh each morning and revoke the previous day's. This cadence is short enough that a leaked token is useful for only a few hours, and long enough that a human can actually work with it without hitting rotation friction every hour. Calendar-based rotation is easy for humans to comply with.

Weekly or on-incident, for interactive dev tokens that must be long-lived. Some tokens cannot be rotated more often — the tooling does not support it, or the provider charges per rotation, or the token is attached to a workflow that would break. For these, narrow the scope aggressively, set a calendar reminder, and rotate on a fixed cadence or immediately on any suspicion of compromise. Never "set and forget." A token with no rotation schedule is a token that will be rotated exactly once: after it has been exploited.

Pre-send secret scanning

A pre-send scanner is a small hook that runs on the prompt text before it leaves your machine, matches against known secret shapes, and refuses to send if it finds anything. It is not a clever system. It is a regex list and a boolean. Its job is to be loud at exactly the moment you were about to do something you would regret.

The patterns that catch the overwhelming majority of real incidents are shape-based:

`AKIA[0-9A-Z]{16}` for AWS access key IDs, `ghp_[A-Za-z0-9]{20,}` for GitHub personal access tokens, `xoxb-[A-Za-z0-9-]+` for Slack bot tokens, `-----BEGIN [A-Z]+PRIVATE KEY-----` for any PEM-formatted private key, `sk-[A-Za-z0-9]{20,}` for OpenAI and Anthropic keys. These patterns have almost zero false positive rate on normal prose and catch the cases that end careers. Integrate the scanner as a git pre-commit hook, an editor save hook, and — most importantly — as a pre-send hook in whatever wrapper you use to call AI APIs. If you cannot find a hook point, write one; twenty lines of Python with `re.finditer` is enough.

MCP server credential model

Model Context Protocol servers present a new and often-mishandled trust problem: each MCP is a separate vendor, running separate code, with its own update channel and its own ability to read every credential you hand it. The wrong mental model is "MCPs are like libraries." The right mental model is "MCPs are like vendors." Each one is a distinct trust boundary. Each one should receive a distinct, scoped token. The tempting shortcut — a single "MCP token" passed to every server in your config — collapses the boundary and ensures that a single compromised MCP leaks every token.

The per-MCP discipline looks like this. For every MCP server you add, you mint a credential scoped to exactly what that server is advertised to do: a GitHub MCP gets a fine-grained PAT for the repositories it is meant to touch and nothing else; a database MCP gets a read-only role scoped to the schemas it queries; a Slack MCP gets a bot token scoped to the channels it is meant to post in. You rotate each of these on a cadence matching its sensitivity. You keep a list of which MCP holds which credential, so when a new CVE drops for a popular MCP server you know immediately which credential to rotate and how much blast to expect.

Treat every new MCP the way you would treat a new SaaS vendor: what data does it see, what credentials does it need, what is the update channel, who runs it, and how do you revoke its access if it becomes a problem. An MCP that cannot answer those questions does not belong in your config.

Rules to remember

- Scope first, rotate second. An unscoped rotating token is still too powerful.
- Per-project `.env` — one file, one scope. Never shared across projects or agents.
- Long-lived admin tokens never go into agent context. Ever.
- Rotate per run for agent runs. Rotate daily for engagement tokens.
- Run a secret scanner over every prompt before it leaves your machine.
- Treat each MCP server as a separate trust boundary. Distinct tokens.