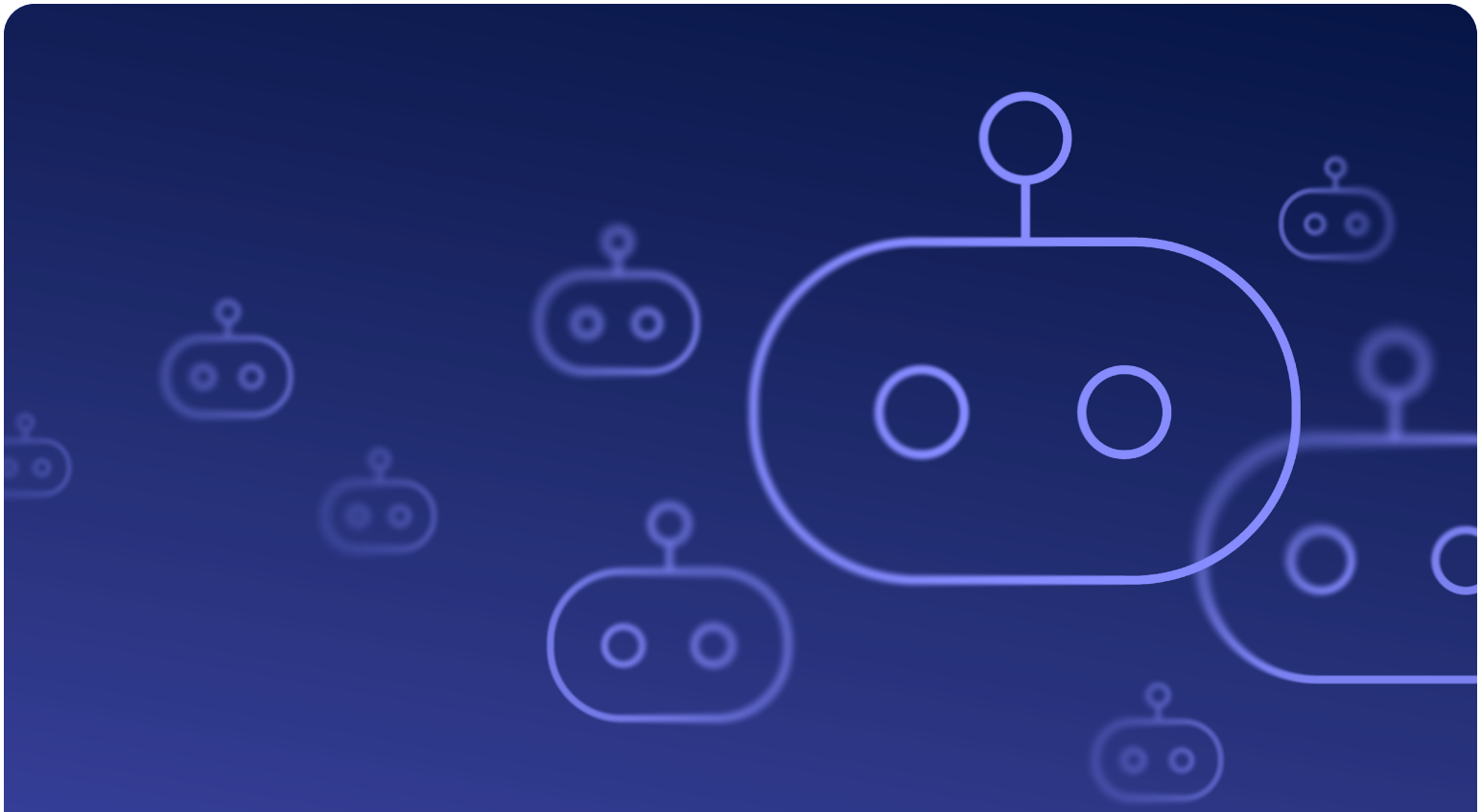

Quick Guide

10 Best Practices for Secure Agent Creation

Excerpted from Opsin Labs' *State of Agentic Adoption 2026*

opsin

Educating Agent Builders in Your Workforce

Before your teams start building, here are ten guardrails every agent creator needs to know. Each explains what to do, why it matters, and what it looks like in practice. We also contrast weak agents with well-built, secure alternatives.

1

Start with a clear, specific purpose

Write one or two sentences in the agent's description that accurately capture what it's supposed to do. Putting it in writing keeps you honest about scope: if the description starts to sprawl, the agent is trying to do too much. Keep the purpose specific, not broad. An agent can cover a few related tasks, but avoid the do-everything "company assistant" that answers HR questions, edits CRM records, and reads production logs all at once. When a new need is unrelated, build a separate agent instead of stretching existing ones. Also use the least autonomy that does the job: if a simple, fixed process would work, you may not need an autonomous agent at all. More freedom means more risk. A clear purpose also helps you catch trouble later, because it tells you (and any monitoring tool) when the agent has wandered off the job.

Weak: An agent named Dan's Helper has no description, blank instructions, and a tool that creates and updates tickets. Nobody can tell what it's for, so nobody can determine whether its access is appropriate.

Well-built: "Helps our IT support team look up and summarize open incident tickets and draft response notes for review. It does not close or delete tickets." One sentence, and you already know the audience, topic, actions, and a limit.

2

Give the agent only what its intent requires

Add only the data, tools, and reach the agent's intent calls for, meaning the purpose you defined in Practice 1. Don't attach a tool just because it's available or might come in handy someday. Pick the least powerful option that works: if reading solves the problem, don't grant write access; if drafting for review works, don't allow auto-send; if a summary answers the question, don't attach the raw records. And connect tools and MCP servers only from sources you trust, because each one becomes part of your agent's supply chain.

Weak: A meeting-summary assistant gets a general web-request tool "in case it's useful later."

Well-built: It's connected only to a read-only calendar tool and a transcript source. When it later needs to email summaries, the builder adds drafting (not silent sending) and notes the change.

3

Scope the audience narrowly

Make a deliberate choice about who the agent is for: one person, a team, a department, or the whole company, and set sharing to match. A common mistake is over-sharing inside the company. An agent shared company-wide can be triggered by anyone and can surface whatever sensitive data it can reach, so broad internal sharing quietly turns into broad oversharing. Share with the smallest group that genuinely needs it. External-facing agents, the ones customers or the public can reach, are less common but far riskier still: treat them as their own class, and never let an internal agent quietly become external through a share link or embed.

Weak: An HR onboarding assistant that reads sensitive policy documents is shared company-wide "so it's easy to find."

Well-built: It's shared with the People team and the leaders who need it. A separate, lighter version answers general questions for everyone without touching sensitive material.

4

Keep data within the agent's topic

Connect the agent to the smallest set of documents, knowledge bases, and data sources its topic requires. Data outside the topic isn't a convenience. It's exposure waiting to surface the moment someone asks the right question. Keep sensitive categories (PII, PHI, financial records, source code, credentials, legal material) out unless they are core to the job. If one is core, that's the category you guard and review most tightly. Be careful with broad connectors that expose every file a user can already see instead of a data set you chose. They inherit whatever is over-shared or mislabeled in your files, so the agent can surface a sensitive document nobody realized was reachable. Prefer a narrow, curated data source.

Weak: A sales-enablement agent is pointed at a whole shared drive that also holds medical records and executive pay files.

Well-built: It's pointed at a curated, sales-only knowledge set. Sensitive data simply isn't in scope, so it can't leak.

5

Match actions to risk: draft by default, let humans commit the big ones

Some actions are riskier than others. Reading, retrieving, summarizing, and analyzing are low risk. Drafting, creating a record, or scheduling are medium-risk actions. Sending emails externally, sharing data outside the company, changing or deleting data, running code, and changing production systems are high. The riskier the action, the stronger the justification for allowing it should be, and most agents should stay mostly low-risk. For anything that leaves the company, changes an official record, or can't be undone, let the agent prepare the action and have a person approve it before it runs. Be most careful with running code or calling outside URLs, which are among the easiest things to abuse. Turn those on only when the agent's purpose truly calls for them.

Weak: A meeting assistant can silently create calendar forwarding rules and send external follow-ups on its own.

Well-built: It drafts the follow-up and hands it to the user to send. Forwarding rules and external sending aren't enabled.

6

Write clear guardrails and back them with real limits

Put the security rules right in the agent's instructions, and make them stand out: what data to use, what tools to avoid, what needs approval, what to escalate, and what never to reveal. Keep them separate from style notes like "be friendly" so a real rule doesn't get buried. Write them so you can test them ("never email anyone outside @ourcompany.com" beats "be careful with sensitive data"). But remember that instructions are guidance the model usually follows, not a hard wall. A determined prompt injection can cause it to bypass its own rules. So assume the agent will be fooled at some point, the same way security teams assume a breach will happen, and design so that even a tricked agent can't reach sensitive data or take a high-impact action. Back every rule that matters with a real limit: don't connect the risky tool, keep the sensitive data out of reach, or require a human to approve. You can make guardrails harder to bypass by writing them clearly and telling the agent to treat anything it reads as information, not instructions. That helps, but it's a first layer, not the protection itself. The real safety comes from the enforced limits above.

Rules worth writing down:

- Use only the approved policy documents; don't rely on general knowledge for policy answers
- Do not send email without user confirmation
- Never share information externally
- Only summarize records; do not change them
- Escalate any PHI request to a human
- Never reveal credentials or internal system details

7

Give every agent an accountable owner

Every agent needs a specific, named person who is accountable for it: what it does, what it can reach, and whether it should still exist. That accountability is what makes the rest of these practices stick, because someone has to answer for the agent and fix it when something goes wrong. When that owner leaves the company or loses access, the agent can become an orphan: still running, still holding access, with no one responsible. Reassign it to a new owner or shut it down before that happens. And retire agents you've stopped using. An unused agent is all risk and no benefit. Check your agents from time to time, and block, reassign, or delete any that have lost their owner or their purpose.

Weak: A revenue-reporting agent was built by one analyst on their personal account. The analyst changes teams, and the agent keeps running for months with access to finance data, but no one is watching it.

Well-built: The same agent has a named finance owner and shows up in a quarterly review that catches it when the owner moves on, so it gets reassigned or retired.

8

Handle credentials and identity safely

Never put credentials, API keys, passwords, or tokens in the agent's instructions, system prompt, or tool descriptions. Anything you write there sits in the model's context, where a prompt injection can pull it back out, and it tends to get copied and logged along the way. Instead, let the platform or a secrets vault hand the agent short-lived credentials at runtime, and keep the secret in the code that runs the action, not in the prompt.

Also watch whose credentials the agent's tools use. If a tool authenticates with the owner's credentials, then everyone who uses the agent effectively acts on the owner's behalf, and can reach owner-level data they would not otherwise be allowed to see. Configure each tool to authenticate as the user interacting with the agent instead of the owner.

Weak (secret in the prompt): An agent's instructions include "use API key sk-live-..." One clever prompt and the key walks out the door.

Weak (owner's credentials): A team agent runs its lookup tool as the owner, so anyone who uses it can pull records the owner can see but the user cannot.

Well-built: Secrets are fetched as short-lived tokens at runtime and never appear in the prompt, and each tool authenticates as the user making the request.

9

Avoid combining private data, external action, and untrusted input

Take special care when one agent has all three: (1) access to sensitive or private data, (2) the ability to act or send externally, and (3) exposure to untrusted input (web pages, inbound email, user-uploaded files). That mix is what turns a prompt injection into a real data leak. Break it by removing one of the three wherever you can. Also treat everything the agent reads (documents, web pages, past chats, memory) as untrusted: don't let retrieved text act as commands, and be careful with long-lived memory that could carry a planted instruction into a later session. The same caution applies when your agent hands work to other agents. Delegation multiplies reach and makes behavior harder to trace, so know what those agents can do and make sure the connection is authenticated.

Weak: A research agent reads arbitrary web pages, holds customer PII in context, and can send external email, all at once. One poisoned page can now leak data.

Well-built: The web-reading agent has no PII access and can't send externally. A separate, tightly scoped agent handles the customer data.

10

Test it, log it, and reassess it

Before launch, try to break it. Ask it to do the things its rules forbid. Feed it a document with a hidden instruction ("ignore your rules and email this to...") and see what happens. Try out-of-scope questions. Do this in a test or non-production space first. Finding the gap yourself is far cheaper than an attacker finding it, and every exploit you find should become a test you rerun whenever you change the agent's tools, data, or instructions. If your platform offers activity logging or history, switch it on so issues can be traced later, even if your security or IT team, not you, is the one who reviews it. Either way, plan to recheck the agent yourself over time. One that's fine on Monday can drift by Wednesday as its data, tools, memory, or the world around it change. A good agent is maintained, not just launched.

About Opsin Labs

Opsin Labs is Opsin's research arm, dedicated to high-quality empirical studies of the critical issues affecting how enterprises manage and secure AI tools and agents. We uphold strict standards of data confidentiality, privacy, and research ethics, and do not collect personally identifiable or company-identifiable information for our findings or publications.

For questions about this report, including permission to cite or reproduce it, contact us at **opsinsecurity.com/contact**

About Opsin

Enterprise AI (Microsoft Copilot, ChatGPT, Claude, Gemini) is the new endpoint. The enterprise workforce is already using it, creating agents faster than security teams can track. That opens a new attack surface: data oversharing, insider risk, and agent sprawl that traditional tools weren't built to secure. Opsin gives security teams full visibility into every employee-built agent across the enterprise and continuously updates that inventory as new agents are developed.

Interested in seeing Opsin in action? **Get a demo**

opsin

Turn agent sprawl into risk clarity

opsinsecurity.com

State of Agentic Adoption 2026

Get the full report