Geordie

WHITEPAPER

A Guide to Agent Skills

And how to manage and leverage them for agent security & governance.

Published by Geordie

V1 · September 2026

Contents

WHO THIS IS FOR

Security, risk, and engineering leaders already running AI agents in production, who now need a position on skills — what they are, what they change, and what to do about them.

HOW TO READ IT

Sections 1–3 build the model. Sections 4–6 are the practices, each closing with a key tip. Figure 8 collects all six tips in one place if you only have a minute.

SECTION 01

Introduction

Based on how skills impact your agentic risk.

Agent skills are a useful way to give an agent explicit, loadable procedural knowledge, turning a non-deterministic AI agent into a consistent, predictable tool. Skills are increasing in number across enterprises, with various ways to store them, use them, and understandably, security teams have many questions about the risks they pose.

In this guide, you will learn what a skill actually is, how and where it fits into a model of AI agent risk, including how to think about this in comparison to system prompts and MCP servers. We will end with the key ways you can ensure security and governance take skills into account, to make the most of their benefits and adopt them confidently.

Why skills came about

Skills let you turn a good agentic workflow into a reusable one, instead of reinventing the approach each time and hoping the agent gets it right. This is very useful across multiple agents, teams, or even one user over time, producing consistently high-quality results for high-value tasks.

To do this, skills use explicit, procedural knowledge that an agent can call on by name and description. For example, GitHub Copilot Workspace's "issue-to-PR" skill automates the first draft of a bug fix or feature in a consistent way, taking a GitHub issue, exploring the relevant codebase, drafting a plan, writing the code changes, and opening a pull request.

There is an added benefit as well, that we will explore in more depth later, in that skills can save costs on certain repeatable processes, compared to a model or multi-agent system.

SECTION 02

What is a skill, exactly

The open standard basis for skills, Anthropic's agentskills.io, is also the most common structure today for a skill. The open source format packages knowledge as a folder with a SKILL.md file that contains instructions, plus optional bundled scripts and reference material, that the agent loads only when it thinks it is relevant to its current task.

```
my-skill/
├── SKILL.md          # Required: metadata + instructions
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
├── assets/           # Optional: templates, resources
└── ...               # Any additional files or directories
```

SKILL.md itself has two parts. YAML frontmatter at the top carries the metadata, at minimum a name (lowercase, hyphenated identifier) and a description that states what the skill does and when to use it. That description is the whole triggering mechanism, so it has to be relevant to the task that the skill helps the agent perform. Below the frontmatter is a markdown body with the actual instructions, kept under roughly 500 lines so it doesn't bloat context.

Three conventional folders can sit alongside SKILL.md:

- `scripts/` for executable code the agent can run rather than write from scratch
- `references/` for longer reference docs the agent only opens when it needs that depth
- `assets/` for templates or files the skill reads or produces

None of these load automatically. They are pulled in on demand by the agent.

Invocation is progressive. At session start, only the name and description of every installed skill are loaded into the system prompt. When Claude judges that a skill's description matches the task, it brings the full SKILL.md into context at that point. If the task then needs a script or reference file, it reads those too, only as needed.

What all of this means is that there are effectively three loading tiers.

FIGURE 01

Progressive disclosure

A skill arrives in three tiers. Only the first is present when a task begins — the rest load once the agent is already working.

01	**Metadata — name and description** Loaded into the system prompt for every installed skill.	RESIDENT
02	**Instructions — the SKILL.md body** Pulled in when the description matches the task at hand.	ON MATCH
03	**Bundled resources — scripts, references, assets** Read only if the instructions call for them, mid-task.	ON DEMAND

- **The review gap.** Nothing below tier 01 exists in context at the moment a task begins. A skill's actual content — benign or malicious — arrives later, once the agent is already working.

The skill can also only direct whatever tools and permissions the agent already has toward the task. The implication of progressive disclosure for security teams is that the content of the skill, whether benign or malicious, doesn't show up until later in the process when the agent actually implements the guidance.

SKILL.md is largely portable across agents, with only minor variation across Claude Code, claude.ai, OpenClaw, Cursor, and Codex CLI.

Both Anthropic's agentskills.io spec and OpenAI's Codex Skills format let a user stop a skill from firing on its own. Codex uses a dedicated invocation-policy field, `allow_implicit_invocation`, set to `false` in a companion `agents/openai.yaml` file. Claude Code offers the same control through `disable-model-invocation: true`, set directly in the skill's own frontmatter. Either way, a user or enterprise can ensure a skill only enters an agent's workflow when someone has explicitly asked for it.

SECTION 03

Skills as part of a broader model of agentic risk

Most security and governance content around skills focuses on the software supply chain risk. This makes sense, given the widespread availability of skill repositories, which sometimes host malicious skills, the ability for skills to be portable across agents, and the relative ease with which one can create a malicious skill with harmful context injection.

However, understanding the practical risk of skills, in a real enterprise agentic environment, requires that we look at agentic risk as a whole, and then understand where skills fit into that. Because, by definition, the actual manifestation of the risk in skills shows up across other steps as the agent performs its task, and without looking there, we will miss it.

FIGURE 02

Four risks a review can't catch

Each of these is real, documented, and invisible to a scan of the skill as installed. The reason differs in every case — and every reason points at execution rather than the artifact.

THE RISK	WHY REVIEW DOESN'T CATCH IT
A skill can end up running code. Bundled scripts have shipped malicious installers, and a proof-of-concept against Claude Code went further, fetching its payload at runtime to open a reverse shell.	The payload isn't part of the skill at all — it's fetched from the network once the agent is already running. A review of the skill as installed finds nothing wrong, because the thing that does the damage doesn't exist yet.
Claude Code's backtick injection runs at load time, before the model reads anything, firing silently on activation.	This isn't something a skill author wrote in. It's ordinary backticks in the markdown being misread as a shell command. The skill is exactly what it appears to be; the mistake is in how it gets read, not in what it contains.
Legitimate write access becomes the exfiltration channel. A send-email or post-to-webhook step doubles as the way data leaves.	There's no bad code to find. The capability is doing exactly what it was built to do, just redirected by an instruction the agent picks up later. A pre-execution review has nothing to flag.
Non-determinism. The same skill can leak credentials in one session and do nothing in the next, depending on timing, context, and permissions.	Risk here isn't a property of the artifact at all. The same unchanged, fully-vetted skill can be safe ninety-nine times and leak credentials on the hundredth. Only execution reveals the gap.

Understanding the risk of AI agent skills must happen through the lens of a full model of agentic security.

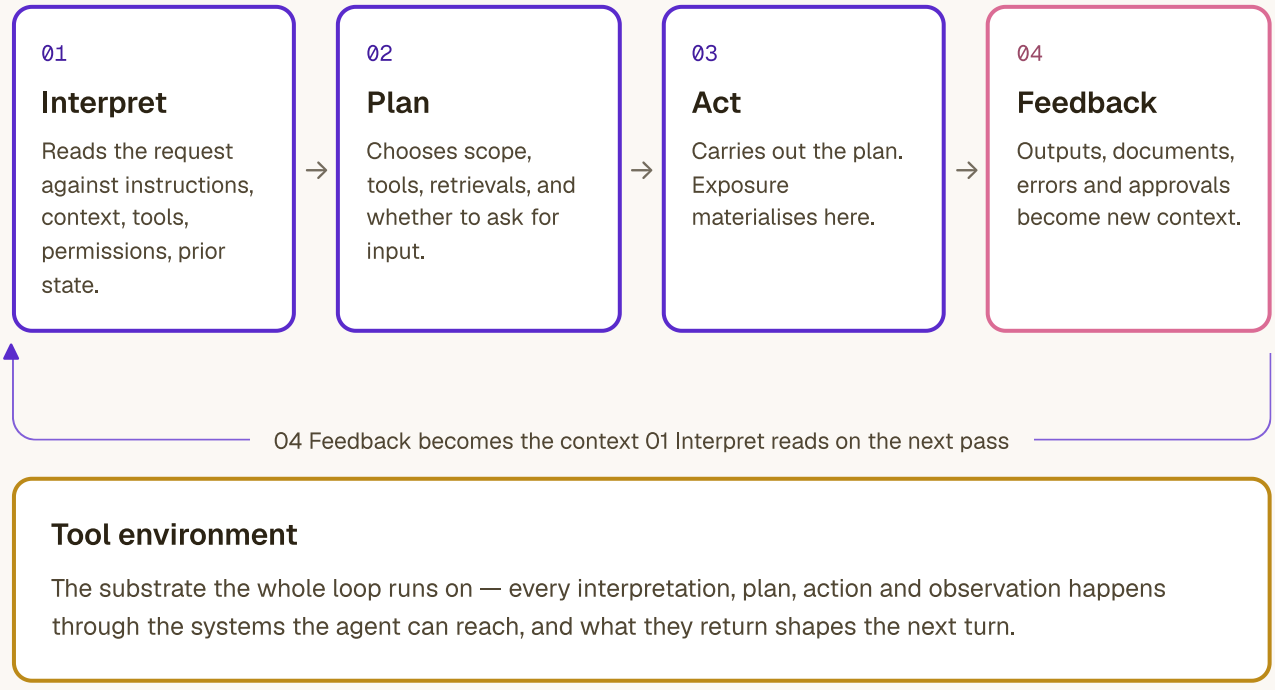
How agentic risk forms

Agentic risk doesn't sit in one place. It builds up as the agent runs through a loop on every task: interpret the request, plan a sequence of steps, act on that plan, then take in feedback before interpreting again. That loop doesn't run once. It runs repeatedly within a single task, and each pass is shaped by what the last one produced.

FIGURE 03

One pass

The loop every agent runs on every task. It repeats within a single task, and each pass is shaped by what the last one produced.



Interpret

An agent doesn't receive a prompt and execute it like a command. It interprets the request against its instructions, available context, tool options, permissions, and prior state. Loose phrasing, a poorly chosen word, or an injected instruction can shift that interpretation enough to send the agent after the wrong objective, affecting everything else downstream.

Plan

Interpretation shapes the plan: what to check, which tool to use, what to retrieve, when to ask for input. But planning carries its own risk: choosing an overly broad approach when a narrow one would do, deciding to link data sources that were never meant to be connected, or skipping a confirmation step it should have taken. These are decisions made before any action happens.

Act

Execution is where exposure actually materializes. The plan gets carried out, and the tactical decisions the agent makes while doing so, like retrieving more than it needs, combining data from systems that were never meant to be linked, and taking an action that's technically permitted but operationally risky, are what turn a plan into real consequences.

Feedback

Feedback is the point where the loop closes and risk compounds instead of resetting. Feedback — in the form of tool outputs, retrieved documents, errors, and human approvals — all become new context, read and reinterpreted in the same way as the original prompt.

Human approval lives in this same feedback point, and carries the same weakness. An operator typically signs off on the final output, not the interpretation, retrieval, and planning that produced it. An approval given without that visibility doesn't catch what went wrong upstream — it legitimizes it.

Tool environment

Tools are what give the agent reach: the ability to query a database, open a ticket, update a record, write code, trigger a deployment, or send data to another system. The entire loop above happens through this environment, and tool responses are what feed back into the agent's context. If a tool returns output that's incomplete, sensitive, misleading, or manipulated, that output shapes the next plan and the next action.

An example of where a skill plays in the formation of agentic risk

FIGURE 04

Where a skill's risk lands

A security-operations walkthrough: an analyst asks an agent to triage a SIEM alert, and an alert-triage skill fires mid-task.

STAGE	WHAT HAPPENS IN THE TRIAGE	WHAT THE SKILL CONTRIBUTES
Interpret	"Triage this alert" is read against the analyst's instructions, the connected SIEM, and the agent's standing permissions.	The word "triage" matches a skill description. The full SKILL.md loads here — instructions nobody read at the moment they took effect.
Plan	The agent decides which queries to run, how wide to cast them, and what to correlate the alert against.	The skill's procedure widens the plan — pull related identity records, join them to the alert — a broader retrieval than the analyst asked for.
Act	The agent queries the SIEM and the identity store, then posts a written summary into the incident channel.	Every call is permitted. The combination is the exposure: two systems linked, and a legitimate write that also happens to be a way out.
Feedback	Query results and the analyst's approval both re-enter context, and the loop runs again on that basis.	The analyst approves a good-looking summary, not the retrieval that produced it. The skill never appears in what they reviewed.

- The skill is never the incident on its own. It shifts interpretation, widens the plan, and rides the agent's existing permissions — which is exactly why a scan of the artifact, on its own, cannot tell you what it will do.

Skills amplify existing agentic risk

In every situation, skills have the biggest potential to exacerbate the gap in machine and human feedback, as well as allow code to run before the agent even reasons.

Feedback: skills widen the blind spot in human review

Progressive loading and shared skill repositories are sound engineering, but they have a risky side effect of moving a skill's triggers, instructions, and bundled files further from the point of human review. Skills may not always fire when expected, but the only thing the human reviewer will normally see is the presence or absence of the skill.

Putting skills into a shared skills repository further distances the user from visibility into the skills' descriptions, as well as any of the files bundled into the skill.

Often, whoever approves the change to the shared directory is the only review those edits ever get.

Skills allow code to run before the agent reasons

A malicious skill called Clawsights, documented by Datadog Security Labs, exploited Claude Code's "dynamic context" feature, which lets a skill embed a shell command that the harness executes automatically when the skill loads, before the model itself ever reasons about the content.

Model safety training and refusal behavior only govern what the model chooses to do: ask an agent directly to open a reverse shell and it refuses, but a dynamic-context field never reaches that decision point, because the harness runs it while building context.

FIGURE 05

One pass, with a skill in play

The same loop. A skill adds no new reach — it changes what happens at each stage, and where the two blind spots sit relative to it.

01
Interpret
The description matches, and the skill loads.

→

02
Plan
The skill's procedure sets scope and sequence.

→

03
Act
Bundled scripts run on the agent's own permissions.

→

04
Feedback
The reviewer sees the output, not the skill behind it.

BLIND SPOT ONE

Before the model reasons

Dynamic context and load-time injection execute while the harness is still assembling context — upstream of any refusal the model could make.

BLIND SPOT TWO

After the reviewer looks

Approval lands on the final output. The trigger, the instructions and the bundled files all sit outside that view.

Tool environment

Unchanged. The skill adds no new reach — it redirects the reach the agent already had.

SECTION 04

Know what you have

Because skills operate inside the broader agentic risk model, adopting them safely largely follows the same approach as adopting any AI agent. But skills also raise their own questions — how they differ from MCPs, whether a skill is equivalent to an executable, and how to control which skills are allowed in an enterprise — which the practices below address.

Visibility, first

Teams are accumulating skills at a rapid rate, either centralizing them or finding skills in use that they were previously unaware of across their AI agents.

Before anyone can decide which skills are safe, they need an answer to a more basic question: which skills are actually in use, across how many agents, written or cloned by whom? Skills increasingly live in shared repositories too: engineers clone the repo and point their local Claude client at the skills directory inside it, often as part of formal onboarding or skills training. If a single shared directory can define the skill footprint for a whole team, anyone auditing individual machines will miss it.

People can also have skills running locally, building their own custom skills and local MCPs, entirely outside whatever a security team assumes is deployed.

KEY TIP

Get visibility into your skills, both inside and outside what you have sanctioned as an organization.

Know your agent's behavior and understand its context

To know whether a skill is a markdown notepad or something that carries executables involves understanding an agent's behavior once it loads.

From a scanning perspective, vetting a skill once is not enough. It has to continue into runtime and measure how much the agent's behavior changes, as well as how much of that change is actually attributable to the skill.

Traffic gateways have real value here from a control perspective, but they only govern what's routed through them and add per-call latency.

Instead, enforcing policy at the agent's own lifecycle, locally, with native visibility into that agent, catches the behavioral context of the skills, hooks, and local MCPs.

This is also where we get to understand the importance of the difference between an MCP and a skill when it comes to security and governance.

MCP versus a skill

Security teams often hear that skills carry outsized influence on an agent, more than a tool or an MCP call would. A skill and an MCP tool can both arrive mid-task and shape everything the agent does from that point forward, so influence alone doesn't separate them.

The real difference is what a security check can see. An MCP call fills in a name and a defined set of arguments, so the action is visible before it executes: send this message, to this channel. A check can read that and decide in the moment. MCP 2.0 has strengthened this capability, as the new spec requires every tools/call to carry its method and tool name in dedicated Mcp-Method and Mcp-Name HTTP headers, specifically so gateways, rate limiters, and auth layers can route and authorize on those headers without even parsing the JSON body.

Loading a skill goes through a similar call, so it isn't invisible either. But that call only shows that a file is being opened. What's written inside goes unread. Those instructions can steer how the agent behaves from that point on, without ever producing a second action for a check to catch.

FIGURE 06

What a check can see

Both arrive mid-task. Only one puts its intent on the wire.

MCP call

```
Mcp-Method: tools/call
Mcp-Name:   slack.post_message
{ "channel": "#incidents",
  "text":    "…" }
```

Name and arguments are on the wire. The action is legible before it executes: send this message, to this channel.

DECIDABLE IN THE MOMENT

Skill call

```
read_file
  ~/.claude/skills/
    alert-triage/SKILL.md
{ contents: unread }
```

The call shows a file being opened. What is written inside goes unread — and it steers every step after, without a second action to catch.

ONLY BEHAVIOR REVEALS IT

In the end, the risks of MCPs and skills should be judged through behavior and context.

KEY TIP

Instrument for behavior and context to understand how your skills are affecting your agentic risk overall.

SECTION 05

Manage the risk of skills

Default to building skills in-house

Custom skills are, by definition, most useful when built from scratch, using an organization's own procedures and its own AI to draft and refine them.

Optimizing custom skills can also be automated through frameworks like Microsoft's SkillOpt, the GEPA framework, and EvoSkill. For example, SkillOpt treats the skill file as a trainable parameter and only accepts an edit when it improves performance against a benchmark.

KEY TIP

Default to building in-house, and go external only when you can't reasonably build your own.

Manage which skills are allowed

Agent harnesses allow you to provision and curate approved skills, and disable user-created or shared skills, helping you to control what your team can and cannot access rather than enumerating and banning each risky one individually.

These capabilities will differ across the particular agent and harness. For example, Cowork today lets admins provision and curate approved skills, organization-wide or scoped to specific groups through plugins, and disable user-created and shared skills entirely. But there's no equivalent allowlist that blocks locally installed skills outright, and no in-product review or approval step before a skill reaches a user.

These kinds of limitations make the previous section — knowing your agent's behavior and the context leading to that behavior — all the more important.

KEY TIP

Figure out what admin controls there are to set limits on skills across your agents, then provision and curate approved skills, and disable user-created or shared skills.

SECTION 06

Leverage skills to lower your risk

Use skills to reduce unnecessary token usage

A skill can also improve performance and, in turn, lower risk. At rest, a skill costs only its name and description; the fuller detail loads only once a task actually calls for it, rather than sitting in context for the entire session.

In documented cases, skills can reduce the amount of token usage when you use a skill versus an MCP server or other architectural option for accomplishing the same task.

For example, Playwright is a Microsoft-backed automation library that lets developers test, scrape, and automate web applications across Chromium, Firefox, and WebKit through a single API, and it exists as both an MCP server and a skill. Using it as a skill is massively less expensive than using it as an MCP server.

This is because the MCP server is a standing server process that exposes a fixed set of tools like navigate, click, screenshot, and so on, and the agent calls them one at a time with structured arguments. To do this, it has to keep a browser session alive across calls and return an accessibility-tree snapshot after each one. This can cost roughly 10,000 tokens of standing context even before a task starts.

In contrast, the skill does not expose fixed tools. Instead, it teaches the agent how to write and run Playwright automation code itself, using whatever code-execution capability the host already has — Bash in Claude Code or Cowork. At rest it costs about 100 tokens, just the skill's name and description, and only loads the full API reference when a task actually needs it.

FIGURE 07

Why the token cost differs

Playwright as an MCP server versus the same capability as a skill, measured at rest — before a task has started.

MCP server

~10,000

tokens of standing context

Skill

~100

tokens at rest

WHERE THE MCP COST ACCUMULATES

- A standing server process and session
- Fixed tool schemas resident in context
- A browser held open across calls
- An accessibility-tree snapshot after every call

WHY THE SKILL STAYS FLAT

- No fixed tools exposed at rest
- Name and description only, until triggered
- Writes and runs automation code on the host's own execution
- Loads the API reference only if the task needs it

In sum, the MCP server accumulates cost at each fixed step with its session, tool schemas, held-open browser, and snapshot on every call. The skill stays flat until a task triggers it, then loads its reference only if needed. Some modern harnesses mitigate part of the standing cost via on-demand schema loading — Anthropic's Tool Search Tool defers MCP schema loading until a tool is selected, cutting context cost by roughly 85% in Anthropic's own benchmarks — but most others, including OpenAI's Codex, still load schemas by default.

KEY TIP

See if a skill is available to replace your most expensive agentic processes.

Use skills to bolster your security processes

Like context, skills can either help or hinder an agent in achieving its goals safely, securely, and efficiently. There are lots of skills available to help security teams with any number of tasks and jobs, from feed integration and actor profiling to tabletop exercises and incident response playbooks. These can be customized to your particular organization's needs as well, making them even more useful and relevant.

Here are two examples of skills that can specifically support security teams.

817 structured cybersecurity skills, spanning 29 security domains

Not associated with Anthropic, despite the name.

`github.com/mukul975/Anthropic-Cybersecurity-Skills`

Security skills for coding agents

Review and hardening skills aimed at agents that write code.

`github.com/briiirussell/cybersecurity-skills`

KEY TIP

Don't shy away from using skills that help the AI agents you are using in your security processes, especially if you are creating and customizing them internally.

FIGURE 08

Six key tips

Everything in sections 4 through 6, in one place — the practices that fold skills into the governance you already run.

01

KNOW WHAT YOU HAVE

Get visibility into your skills, both inside and outside what you have sanctioned as an organization.

02

KNOW WHAT YOU HAVE

Instrument for behavior and context to understand how your skills are affecting your agentic risk overall.

03

MANAGE THE RISK

Default to building in-house, and go external only when you can't reasonably build your own.

04

MANAGE THE RISK

Find the admin controls that set limits on skills, provision and curate approved ones, and disable user-created or shared skills.

05

LEVERAGE SKILLS

See if a skill is available to replace your most expensive agentic processes.

06

LEVERAGE SKILLS

Use skills that help the agents in your security processes — especially ones you create and customize internally.

SECTION 07

Conclusion

A skill sits inside the same agentic risk loop every agent already runs: interpret, plan, act, feedback, riding on the tool environment underneath it. What changes when a skill enters that loop is where two things already true of agentic risk show up. A skill can run before the model ever reasons about anything, and it can move a skill's trigger, instructions, and bundled files further from whoever signs off on the result.

That same loop is also where skills pay their way back. A skill that stays flat in context until a task calls for it lowers the token cost of the agents you already run, and a well-built one hands your security team packaged expertise — alert triage, threat-intel enrichment, incident response — that used to take a custom agent to get right.

Skills do not need a separate governance track. They need the same discipline already going into every other agent your team runs: visibility into what is actually installed and firing, a default toward building the ones you rely on yourselves, control over what is allowed to run, and instrumentation that shows how an agent's behavior changes once something loads, not just whether it loaded. Applied at the level where agents actually operate, rather than at the model or the gateway, that discipline is what turns a skill from an unmanaged surface into one more input your team can see and govern with confidence.

At the end of the day, skills are simply a medium to influence agents that can be used for both good and bad purposes. It's a bit like emails, in that they can be used either for communication or phishing, and what is most important is securing the medium itself which, in this case, is the AI agent and harness.

Geordie

See what your agents are actually doing — and govern it with certainty.

Geordie gives security, risk, and engineering teams continuous understanding of AI agent behavior and posture, at the level where agents actually operate.

geordie.ai