WHITEPAPER · 2026

Beyond the MCP

Three Foundational Principles for Governing the Agentic Toolset

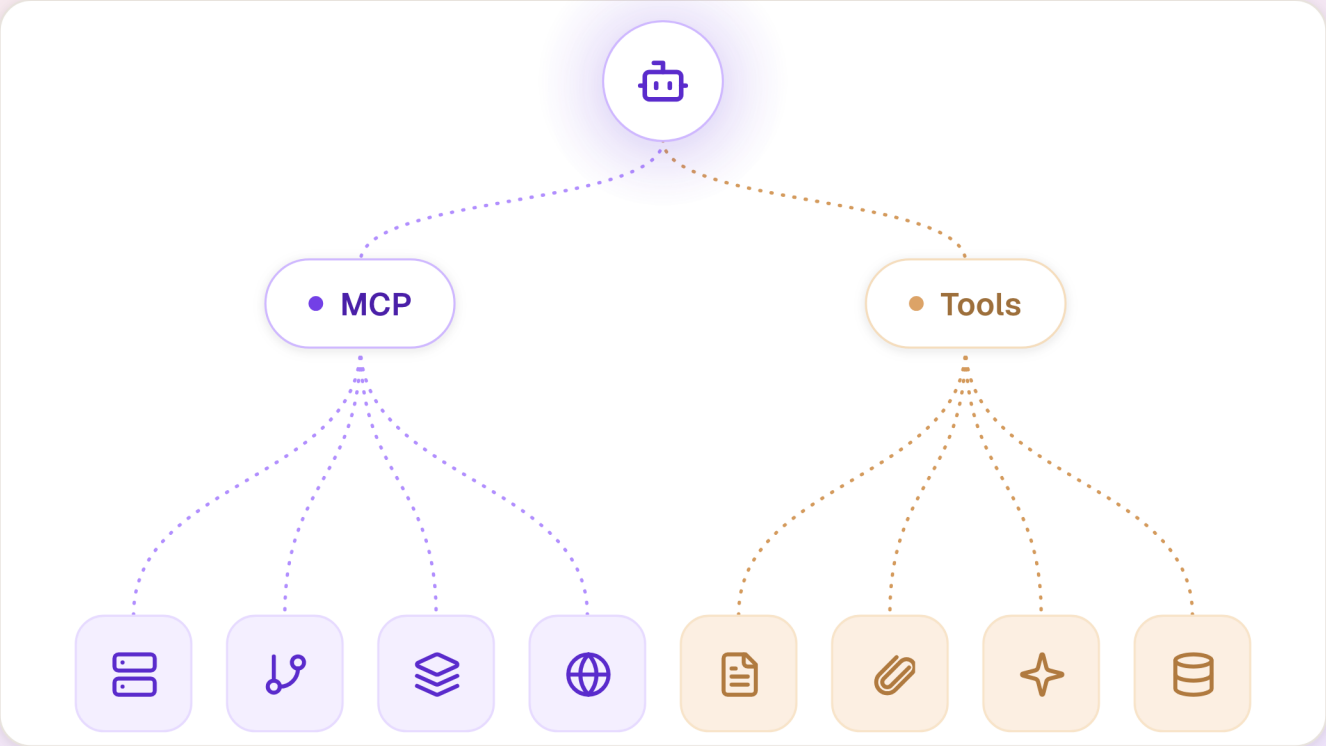
01

Introduction

A lot of the agent security conversation has narrowed around the Model Context Protocol, or MCP. The attention makes sense. MCP is an open standard that gives an agent one common way to connect to external tools and services, and that shared interface creates a clearer place to think about permissions, routing, and enforcement.

The problem is that MCP only represents one part of the tool environment agents use in practice.

Enterprise agents are built with many different kinds of tools and knowledge sources. Some use MCP servers, like a GitHub server for managing issues and pull requests. Others call APIs directly, like calling Stripe to issue a refund. Other tools are hard-coded, wired straight into the agent, like a shell or terminal command that lets it run things directly on the machine. Browser extensions or plugins, like a Chrome extension, and direct connections to SaaS applications operate alongside agents, while other tools retrieve context from databases or documents. Still others operate through skills that package several capabilities behind a single interface.



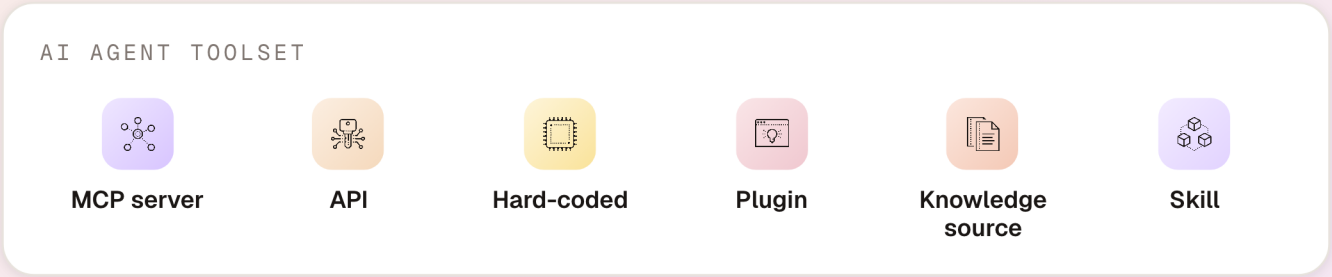
In this guide, you will get a clear picture of the different kinds of tools your agents use, why the broader risk reaches well past MCP, and the three foundational principles for governing the whole toolset.

02

The tool types agents actually use

Knowing what counts as a tool is the first step to knowing where the risk is.

Geordie defines a tool as the mechanism through which an agent interacts with real-world systems. These tools are not interchangeable. Each one changes what the agent can see and do in a different way, and each becomes available to the agent through a different path, so it is worth taking the main types one at a time.



MCP server

A standardized connection, built on the Model Context Protocol, that exposes a set of external tools or services to an agent through one common interface. Rather than wiring up each service in its own bespoke way, the agent calls whatever the server advertises through the shared protocol. A single server can sit in front of many underlying capabilities, and the list of tools it offers can be defined, extended, or changed by whoever operates it, sometimes without the calling team noticing.

API

A direct programmatic connection to a system of record, transaction workflow, or internal service. It is the most familiar tool type to most teams, because APIs have governed software-to-software communication for years. A finance agent might call an ERP through an API to post an invoice, and a procurement agent might use one to check a supplier record or submit a purchase order. In each case the agent acts directly against a production system, with exactly the reach the credential behind that call allows.

Hard-coded tool

Wired straight into the agent's own logic: a shell command, a file operation, or a function a developer built directly into the agent. Because it lives inside the agent rather than behind an external connection, it rarely shows up when teams map their integrations from the outside. It is also often the most powerful tool an agent has, since a local command can read, write, or execute on the machine the agent runs on.

Plugin or extension

A software module that adds capability to an application the agent works through, most often the browser. A browser extension, for example, can read the current page, fill in forms, or click on the user's behalf. Because it sits inside the user's own environment and session, it can see and act on whatever that environment exposes, which is frequently far more than the task in front of it requires.

Knowledge source

Content the agent reads for context rather than a system it acts on: a document, a shared drive, a database, an email thread, a support ticket, or a specific webpage. A sales agent reading an account's notes before drafting an email is using one. It is tempting to treat these as passive inputs, but what they return can change how the agent interprets its task and what it decides to do next, which makes them part of the tool surface even though nothing is executed.

Skill

A packaged capability that presents to the agent as a single action while orchestrating several steps underneath: retrieval, transformation, other tool calls, prompts, and decisions. The agent, and often the person supervising it, sees one clean action, while the real work happens inside. That abstraction is convenient, but it also means a single approval can stand in for a whole chain of activity that is not visible from the outside.

Taken together, these tools give an agent a wide and uneven reach into real systems, and that reach is rarely fixed. The set of tools in front of any given agent changes as work happens, which is where governing them gets genuinely difficult.

03

Tools and agents evolve dynamically

For agents, a changing toolset is normal, not an edge case, and that is a real departure from how software has worked until now. In conventional software, the tools and integrations a system depends on are defined ahead of time. They are declared in code and dependency manifests, pinned to specific versions, reviewed through build and deployment processes, and changed only through controlled release cycles. A security team can read the manifest, see exactly what the system is able to reach, and trust that the answer will be the same tomorrow as it is today, because nothing changes without a release that someone signed off on.

Agents operate in a far more fluid environment. The set of tools in front of an agent is often assembled at runtime rather than declared in advance, and it shifts with the user, the session, the workspace, the tenant, policy, retrieved context, and configuration. The same agent can finish one task with one set of tools and begin the next with another, with no code change and no release in between. Often there is no single manifest to read, because the honest answer to what this agent can do depends on the state of the workflow at that exact moment.

AT DEPLOYMENT

Some of that variation is there from day one. The same procurement agent, deployed to two business units, may be connected to two different sets of tools: one with write access to the ERP to raise purchase orders, the other limited to read-only supplier lookups. Same agent, same prompt, a different tool surface.

AT RUNTIME

Other changes happen while the agent works. A sales agent that starts with read-only CRM access can be granted a quote-generation tool once a deal qualifies, and an agent that writes its own skill gains a capability no one provisioned in advance. The set of tools expands based on what the work requires, not only on what was granted at the start.

So a tool cannot be assessed once and filed away. What an agent can do through its tools is a moving target, which is exactly why governing any single tool in isolation falls short.

04

Governance for tools must be through the lens of the agent

Knowing that an agent can access a tool is only the starting point. Teams also need to understand when that tool becomes available, what inputs the agent sends to it, what context the tool returns, how that response changes the agent's plan, and what further actions become possible as a result.

Gateway approaches can help with the traffic that passes through them, especially where teams have standardized tool access through MCPs, APIs, or a specific proxy path. They can support routing, policy checks, and visibility for those tool calls. The limitation is coverage. Gateways only govern the tool types and workflows that are routed through them. They may give security teams visibility into MCP or API traffic, while leaving hard-coded tools, plugins, extensions, skills, SaaS-native actions, local tools, knowledge sources, and direct agent-framework integrations outside the same model. That creates partial visibility and partial control: teams can see one governed pathway, but still lack a full view of the tools and context shaping agent behavior across the wider environment.

The wider set of tools remains broader than any single gateway path. Even within MCP-based environments, the governance question extends beyond the protocol. Security teams still need to understand the capability being exposed, the data being returned, the agent context around the invocation, and the way the output influences the next step.

> **Skills make this especially clear.** A skill may look like a clean abstraction from the outside. Inside, it may involve retrieval, transformation, tool calls, permissions, prompts, and actions across multiple systems. Governing the entry point gives teams only part of the picture. Understanding the skill requires visibility into what it enables the agent to do and how it shapes the workflow.

> **Knowledge sources deserve the same scrutiny.** Agents use them as context for decision-making. A document, ticket, webpage, database result, email, tool response, or internal knowledge base can influence how the agent interprets a task and what it decides to do next. Treating knowledge sources as passive inputs leaves a major part of the tool environment outside the governance model.

05

The three foundational principles

Taking into account the large variety of tool types in use, and the dynamism of how agents affect tools and how tools affect agents, holistic governance of your agentic toolset comes down to these three key principles.

INVENTORY

01

Start with the full inventory of capabilities around the agent, including what you may not be aware of.

You may not know the full variety of tools underneath your agent: MCP servers, APIs, hard-coded tools, plugins, extensions, skills, SaaS integrations, knowledge sources, and internal systems. From there, teams can assess what each capability exposes, how it is invoked, what context it returns, which agents can use it, and how usage changes across workflows.

CONTINUITY

02

Focus not just on tool posture, but on the continuous connections between tools and agents.

This cannot be only at onboarding or use-case approval. It has to be the same continuous monitoring applied to agent inventories and runtime activities. An agent vetted during initial onboarding can completely change its architecture, capabilities, and risk profile in a matter of minutes by simply adding new tools or writing its own skills. And the reverse is true too: a tool can be a risk or not, depending on its connections and instructions to the agent.

CONTEXT

03

Take control by working with the capabilities that are actually available, between tools and their agents, across the real operating environment.

That includes the tools the agent can call, the knowledge it can retrieve, the skills it can invoke, and the changes that happen as workflows develop. This is the level of context security teams need to deploy with confidence as agents scale in production.

06

Conclusion

The majority of the governance challenge sits in the wider set of tools, knowledge sources, and dynamic capabilities that agents use to get work done. Governing tools effectively is synonymous with governing agents effectively, and it requires a holistic approach that covers the continuing evolution of agentic workflows.

Get in touch with the Geordie team today for a holistic, continuous approach that includes, but also goes beyond, MCPs, so you can move faster with AI agents and their entire toolsets.

Book a Demo →