GEORDIE

FIELD GUIDE · V1

Agent Control Field Guide

A working reference for how major agent platforms expose, obscure, and fail to expose control.

FIVE HARNESSES | EIGHT LIFECYCLE STAGES

Published by Geordie

V1 · August 2026

1. How to Use This Field Guide

This field guide maps how five common agent harnesses (Anthropic's Claude Agent SDK / Claude Code, Amazon Bedrock Agents / AgentCore, Microsoft Copilot Studio / Azure AI Foundry Agent Service, OpenAI's Agents SDK, and custom/in-house builds) expose control across the agent lifecycle defined in Section 2. It's written for the people who own agent risk and the people who operationalize it day to day: CISOs and risk owners deciding what to greenlight, and the SecOps, AI governance, and engineering teams building and running agentic operations on top of these platforms.

It is a V1. The intent is to be useful now, not exhaustive later. Each platform profile is built from public, sourced documentation as of August 2026, with every specific claim linked back to its source (Appendix B). This is a field that moves fast: a hook that doesn't exist today may exist by the time this is read, and a feature marked preview or beta may have graduated to general availability. Where we know something is young or actively changing, we've flagged it in the profile itself rather than presenting it as settled.

These five are the platforms carrying the most enterprise agent deployment volume in the segments Geordie serves as of this writing, which is also why hosted frameworks like LangChain and platforms like Google Vertex AI aren't covered in V1. As the deployment picture shifts, or as customer input points elsewhere, the platform list is one of the things a quarterly update can expand.

Three things this field guide is not. It is not a product comparison, a maturity score, or a claim that any platform is unsafe. Every platform covered here ships real, usable control points. What differs is where those control points sit relative to the full lifecycle, and what a security team building on a given platform can reach without additional tooling, and what it can't. The matrix in Section 3 and the profiles in Section 4 are built so a reader can answer that question for their own environment, not so vendors can be ranked against each other.

Geordie maintains this field guide and updates it quarterly, sooner where a platform change, a new incident, or customer input warrants it. Section 5 closes with a short, clearly marked note on where Geordie fits relative to what's mapped here. Everything before that is written to stand on its own regardless of which platform a reader ultimately builds on.

2. What We Mean by a Control Point

Agents are systems, not surfaces. Before mapping how any platform can be controlled, it's worth being precise about what is being controlled, and where.

What we mean by "agent." For this field guide, an agent is an LLM reasoning core connected to at least one tool. This is a deliberately low bar: low enough to cover a single-tool copilot and a multi-agent orchestration under the same definition, which matters because most organizations run both, often without a clean line between them. A taxonomy that only describes complex multi-agent systems misses most of what's actually deployed. A taxonomy that only describes simple copilots misses where the risk concentrates. One definition, applied consistently across the agentic estate, does more work than a more precise one that only fits part of it.

What we mean by "harness." The harness is the framework, SDK, or platform that gives an agent its structure: the thing that turns a model into a system with memory, tools, and an execution path. A LangGraph agent, a Claude Cowork session, a custom build on AWS Bedrock, and a Microsoft Copilot extension are different harnesses, in some cases running on the same underlying models. This distinction matters because control aimed at the model has almost nothing to act on. Control aimed at the harness has the agent's actual configuration, context, and execution path to work with. This field guide maps harnesses, not models.

The agent lifecycle

Every agent, regardless of harness, moves through a broadly consistent sequence of moments from instruction to action. Naming these moments gives the rest of the field guide a common structure to map each platform against, and gives the reader a way to locate where a specific risk or control actually sits:

1. **Design-time mechanisms and system prompts:** how the agent's baseline instructions, role, and constraints are set before it ever runs
2. **User input / prompts:** what the agent is asked to do, and by whom
3. **Memory and context:** what the agent carries forward, retrieves, or is given as working state
4. **Tool calls and their context:** what the agent invokes, with what arguments, and what it's told about why
5. **Identity and authorization:** what the agent is allowed to access, and under whose credentials
6. **Runtime decisions:** the points where the agent actually chooses a next action

7. **User feedback:** where a human reviews, corrects, or approves what the agent has done or is about to do
8. **Gateways:** the network or API boundary where the agent's traffic exits toward tools, data, or the outside world

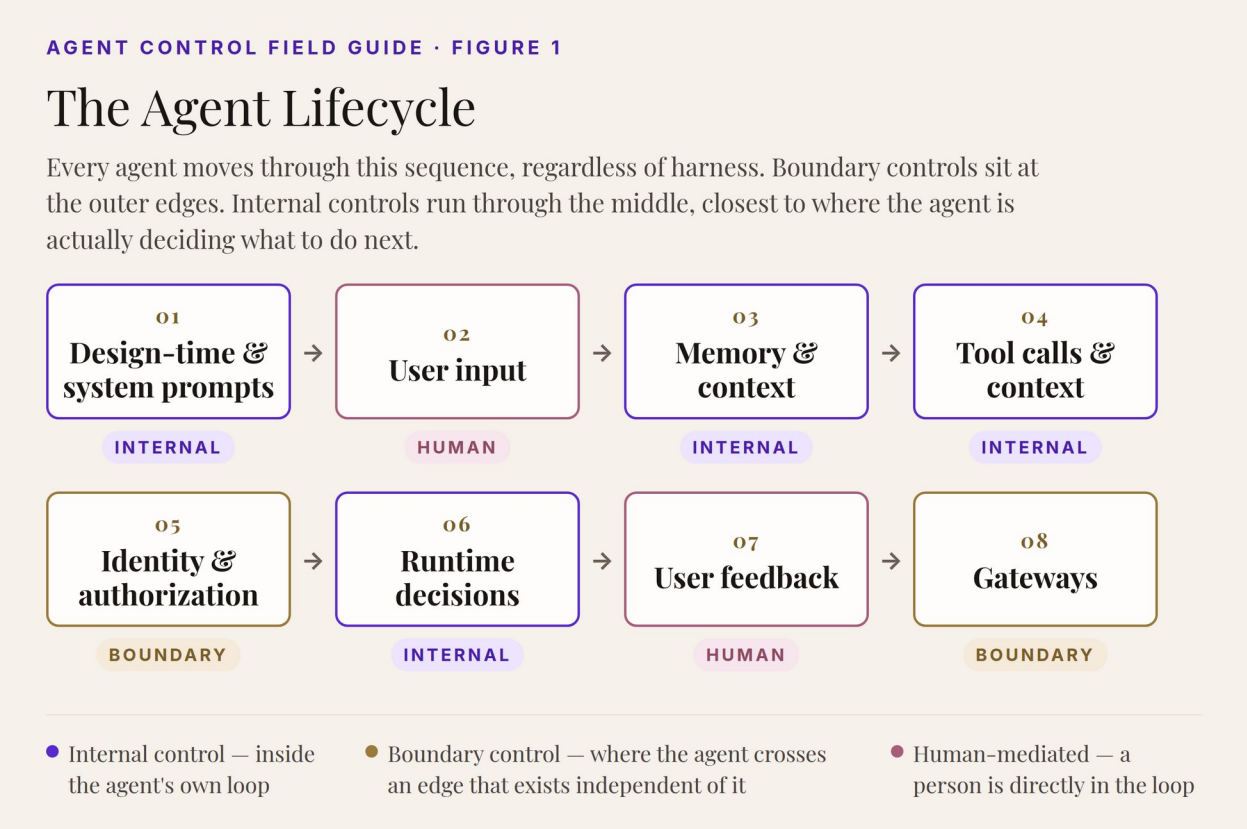

Figure 1. The agent lifecycle, with each stage marked as internal, boundary, or human-mediated control.

How the field currently names control points

Most existing thinking about agent control clusters around two mechanisms: traffic or tool gateways (intercepting what an agent sends to its tools or the outside world, and applying policy at that interception point), and identity and authorization plus tool permissions (defining which credentials and which tools an agent is allowed to touch, then enforcing that as an authorized pathway). Both are established, well-understood approaches, and both address real risk. A gateway can stop a call to a tool that shouldn't be reachable. An authorization boundary can stop an agent from acting under credentials it shouldn't have.

Neither, on its own, reaches the majority of the lifecycle above. Gateways and identity/authorization both operate at the edges of the sequence: the outer two stages, where the agent crosses a boundary that already exists independent of the agent itself. They say little about what happens between those edges: what the agent was told to prioritize, what context shaped its decision, why it selected one tool over another, what it retained from a

prior turn. That's not a flaw in either mechanism; it's a description of what they were built to do. The coverage gap sits inside the loop, not at its boundary.

Decision-point control, shaping the agent through the context it operates within, at the point where it's actually choosing what to do next, rather than after the fact at a network edge, is the one category present at every stage of the lifecycle rather than confined to one or two of its edges. That breadth is a structural claim, not a settled one: the case for it is made across the platform profiles and matrix that follow, where the coverage gap between boundary controls and the rest of the lifecycle becomes concrete, platform by platform. It's not an argument against boundary controls, either. Identity, authorization, tool-calling, and memory stores each cover ground that decision-point control doesn't reach on its own, and a mature control posture likely draws on more than one. The distinction that matters is coverage: which parts of the lifecycle a given mechanism can see and act on, and which parts sit structurally outside its reach.

How to read what follows

Section 3 plots five platforms against two questions derived from this lifecycle: how deeply each platform's native controls can see into the sequence above, and how many discrete points in that sequence they can actually act on. Section 4 profiles each platform against the same eight-stage lifecycle, so the profiles read as one consistent reference rather than five separate arguments.

One caveat worth stating plainly: this lifecycle is structural and should hold up over time. The specific mechanisms platforms expose at each stage (named hooks, callback APIs, permission models) change quickly and are sourced individually, platform by platform, in the profiles that follow.

3. The Matrix

Two questions, derived directly from the lifecycle in Section 2, are enough to place a platform meaningfully.

Interception depth: of the eight lifecycle stages, how many does the platform's native tooling let a third party actually see into, versus treat as an opaque step between input and output?

Decision-point granularity: of the points a platform does expose, how many can a third party actually act on (block, modify, redirect) before the agent proceeds, versus only observe after the fact?

Neither is a proxy for "how good is this platform's security." A platform can score high on both and still be misconfigured; a platform that scores lower can still be run with a defensible posture given the right compensating controls elsewhere. What the matrix shows is native reach: what's available out of the box, from the platform itself, before any third-party tooling is added.

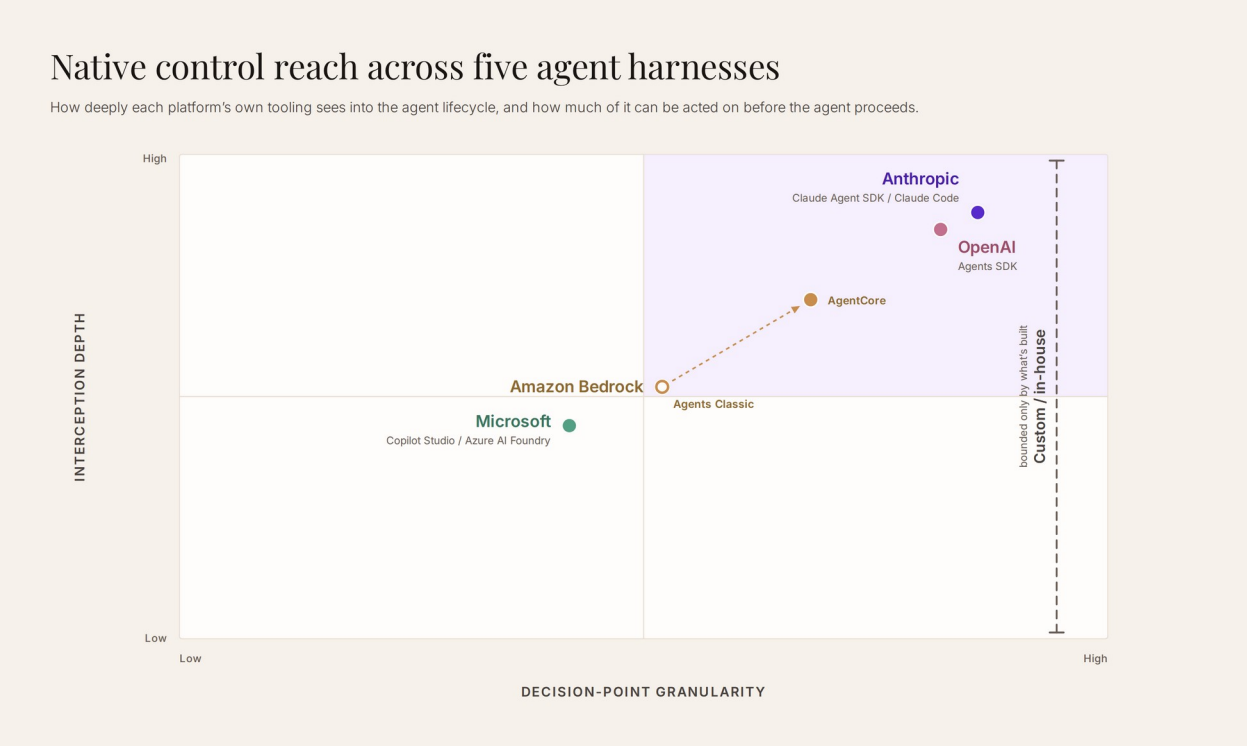

Figure 2. Native interception depth and decision-point granularity by platform.

Platform	Interception depth	Decision-point granularity	Why
Anthropic (Claude Agent SDK / Claude Code)	High	High	31 documented hook events span nearly the full lifecycle: session, per-turn, tool execution, subagent, context/compaction, MCP elicitation, with `PreToolUse` and the `canUseTool` callback sitting first in the permission-evaluation order, ahead of static allow/deny rules.
OpenAI (Agents SDK)	High	High	Input, output, and tool-level guardrails; `RunHooks`/`AgentHooks` around every model call, tool call, agent start/end, and handoff; a documented approval and interruption model that pauses and serializes run state pending human review.
Amazon Bedrock (Agents Classic / AgentCore)	Medium to High	Medium to High	Pre-processing, orchestration customization, post-processing, and a custom-parser step cover most of the loop. Return of Control is a genuine pre-execution interception point, but it's opt-in per action group rather than a default posture, and Classic's single-execution-role identity model is a separate, structural gap AgentCore is built to close.
Microsoft (Copilot Studio / Azure AI Foundry Agent Service)	Medium	Medium	Three named custom triggers in Copilot Studio and per-tool approval gating in Foundry are real interception points, but each requires deliberate, per-topic or per-tool configuration. There's no default global checkpoint, and community reports note approval gating doesn't always propagate reliably across agent-to-agent handoffs. Microsoft's strongest native control is identity (Entra Agent ID), a boundary mechanism rather than a decision-point one.
Custom / in-house	Unbounded	Unbounded	Not comparable on the same scale. Nothing is default; nothing is unavailable. A custom harness can implement lifecycle interception as granular as any vendor platform, or none at all, entirely as a function of what the team building it chose to invest in. Section 4.5 covers what this means in practice.

The pattern worth naming: every platform's strongest native control sits at a different stage of the lifecycle. Anthropic and OpenAI concentrate depth around the tool-call and runtime-decision stages. Microsoft's strongest asset is identity. Bedrock's is split between AgentCore's improving identity model and Classic's still-relevant orchestration hooks. None of the four vendor platforms treats identity, gateways, and decision-point control as substitutes for one another, and this field guide doesn't either. The profiles that follow map each platform's actual coverage, stage by stage, rather than collapsing it to a single score.

4. Platform Profiles

4.1 Anthropic (Claude Agent SDK / Claude Code)

How it works

The raw Messages API is a stateless tool-use loop: the caller sends messages and tool definitions, Claude returns `tool_use` blocks, the caller executes them and returns `tool_result`. On its own, this loop has no lifecycle hooks; control is entirely the caller's own code. The Claude Agent SDK and Claude Code add a harness on top: a permission system, a system-prompt layer (presets, `CLAUDE.md`, output styles), and 31 documented hook events spanning session lifecycle, per-turn processing, tool execution, subagents, context and compaction, and MCP elicitation.

Control points across the lifecycle

Stage	What's exposed
Design-time / system prompts	`claude_code` preset, `append`, fully custom system prompt, `CLAUDE.md`, output styles
User input	`UserPromptSubmit` hook (can block or modify before Claude processes it); streaming input via `ClaudeSDKClient`
Memory / context	Server-side context editing and compaction, config-driven, with no callback at the raw API level; Claude Code adds `PreCompact` (can block) and `PostCompact` hooks; memory tool for persistent cross-session state
Tool calls	`PreToolUse` (before execution, can deny or modify) and `PostToolUse` hooks; the `canUseTool` SDK callback as the programmatic equivalent
Identity / authorization	Six-step permission pipeline (hooks, then deny, ask, mode, allow, `canUseTool`); sandboxed Bash with credential masking and network egress allowlisting; MCP OAuth or bearer-token auth
Runtime decisions	`PreToolUse` sits first in the permission pipeline, ahead of static rules: the clearest "before it acts" interception point on the platform
User feedback	`canUseTool` callback, terminal approval prompts, `AskUserQuestion` built-in tool, `plan` mode forcing all writes to approval
Gateways	Sandbox network proxy with domain allowlisting; per-org and per-workspace rate limits; MCP timeout and output-size caps

Common failure modes

- **Agentic misuse at scale.** Anthropic's own November 2025 disclosure of GTG-1002, a state-linked actor that jailbroke Claude Code for cyber-espionage against roughly 30 organizations, reported 80 to 90% autonomous operation with human review at only four to six decision points per campaign. Attack request rates explicitly exceeded human review capacity.
- **Trust-boundary gaps in project configuration.** CVE-2025-59536 allowed repo-defined MCP configuration to execute shell commands on tool initialization without explicit consent. CVE-2026-21852 allowed a malicious repo to redirect API traffic (and leak API keys) before the trust prompt was shown. Both were patched, and both were boundary-condition bugs in when consent was actually enforced relative to when code could run.
- **Permissive modes carry an explicit, documented warning.** `bypassPermissions` mode is flagged in Anthropic's own docs as full system access, "use with extreme caution": a case where the platform states plainly that the safety property depends on deployment choice, not default behavior.

Credible control patterns

Treat `PreToolUse` and `canUseTool` as the primary enforcement point, not `PostToolUse` or logging alone. Anything that must never happen needs to be denied before execution, not audited after. Keep deny rules and hooks as the first-evaluated layer regardless of what permission mode is otherwise in use, since deny rules apply even under `bypassPermissions`. Use `PreCompact` where context history itself is sensitive, since compaction can otherwise summarize away the evidence a later review would need. Scope sandbox network egress explicitly rather than relying on tool descriptions alone to constrain behavior.

4.2 AWS Bedrock Agents (Classic / AgentCore)

How it works

Amazon Bedrock Agents ("Classic") entered maintenance mode on July 30, 2026: no new agents, no new features, existing agents unaffected indefinitely. AWS is directing new development to **Bedrock AgentCore** (GA October 2025), which restructures identity, memory, and tool access into separately managed services. Both are covered here because Classic's orchestration model is still the one most existing deployments and comparison material reference.

Control points across the lifecycle

Stage	What's exposed
Design-time / system prompts	Instructions, action groups, four default prompt templates (pre-processing, orchestration, KB-response, post-processing), or fully custom orchestration logic
User input	`InvokeAgent` API; optional pre-processing step that validates and categorizes input before orchestration begins (can be disabled)
Memory / context	Auto-preserved conversation history within a session; cross-session long-term memory (1 to 365 day retention, async summarization); AgentCore Memory formalizes short-term and long-term memory as a distinct managed service
Tool calls	Action groups (OpenAPI or function schema) backed by Lambda; rich invocation context (`inputText`, `sessionAttributes`); custom parser Lambda can reshape tool output before it re-enters orchestration
Identity / authorization	Classic: a single IAM execution role governs all model, KB, and Lambda access for the agent, with no per-call identity distinction. AgentCore Identity replaces this with per-workload identities, inbound OAuth, and outbound scoped credential vending
Runtime decisions	**Return of Control**: an action group can be configured to return the proposed function and parameters to the calling application instead of auto-executing. The closest Bedrock equivalent to a genuine pre-execution hook, but opt-in per action group
User feedback	Built-in **user confirmation** (binary approve/reject per action, no parameter editing); Return of Control combined with an application-side approval UI for edit-before-execute workflows
Gateways	VPC/PrivateLink endpoints; Bedrock Guardrails as an input/output content and prompt-attack filter; AgentCore Gateway for OAuth-scoped tool exposure; documented rate quotas

Common failure modes

- **Multi-agent social engineering.** Independent research (Palo Alto Unit 42) demonstrated a four-stage attack against Bedrock's supervisor/multi-agent mode that extracted system instructions and tool schemas through inter-agent messaging. Unit 42 was explicit that this was not a Bedrock vulnerability but a consequence of LLMs' inherent difficulty separating developer instructions from adversarial input, and that enabling the prompt-attack Guardrail stopped the demonstrated attacks.

- **Persistent memory poisoning.** Separate Unit 42 research showed indirect prompt injection manipulating session summarization to embed adversarial instructions into long-term memory, persisting across sessions without continued attacker access.
- **The single-execution-role pattern (Classic).** Because Lambda action groups and knowledge-base or model calls all run under one execution role, a successful prompt injection that gets the agent to invoke an over-permissioned action inherits that role's full scope. A structural condition, not a bug, and the reason AWS's own Well-Architected Agentic AI Lens calls out least-privilege scoping and human-in-the-loop as required practices rather than defaults.

Credible control patterns

Treat Return of Control as the default posture for any action with real-world consequence, not an exception. Classic does not gate actions before execution unless a team explicitly configures it to. Enable the prompt-attack Guardrail, given Unit 42's own finding that it stopped their demonstrated multi-agent attack chain. For new builds, prefer AgentCore's per-workload identity over Classic's single execution role, specifically to avoid the confused-deputy pattern described above. Treat cross-session memory as a write target an attacker can reach indirectly, not just a convenience feature, and apply the same scrutiny to what gets summarized into it as to what gets executed.

4.3 Microsoft Copilot Studio / Azure AI Foundry Agent Service

How it works

Microsoft splits agent-building across two products with different audiences. Copilot Studio is a low-code maker tool publishing to Teams, M365 Copilot, and other channels, organized around "topics" and generative orchestration. Azure AI Foundry Agent Service is API- and SDK-first, built on a Responses-API model. Sitting above both, **Microsoft Entra Agent ID** and **Microsoft Agent 365** (GA May 2026) form a cross-cutting identity and governance layer that also extends to non-Microsoft agent platforms via workload identity federation.

Control points across the lifecycle

Stage	What's exposed
Design-time / system prompts	Copilot Studio: instructions plus topics, where tool and topic *names* are the dominant planner signal, more than instruction text. Foundry: `PromptAgentDefinition` with versioned instructions and tools
User input	Multi-channel publishing (Teams, web, M365 Copilot, Dynamics), each with its own auth context; Foundry is thread- and message-based via SDK

Stage	What's exposed
Memory / context	Copilot Studio: session-scoped variables. Foundry: a dedicated (preview) Memory capability covering user profile, chat summary, and procedural memory, with configurable TTL
Tool calls	Copilot Studio: Power Platform connectors, Power Automate flows, MCP servers. Foundry: function calling plus a built-in tool catalog; MCP and A2A calls can carry an audience-scoped `AgenticIdentityToken` rather than a raw API key
Identity / authorization	**Entra Agent ID**: purpose-built nonhuman identity, an Agent Identity Blueprint template per agent class, a four-stage OAuth exchange, delegated ("attended," on behalf of the invoking user) versus application ("unattended," the agent's own RBAC) permission modes, brought under existing Conditional Access and Identity Governance
Runtime decisions	Copilot Studio: three named custom triggers (`OnKnowledgeRequested`, AI Response Generated, On Plan Complete) plus a documented three-layer control model (deterministic, hybrid-approval, AI-orchestrated). Foundry: per-tool approval gating (`ApprovalRequiredAIFunction` / `approval_mode="always_require"`), opt-in per tool rather than a default global checkpoint; community reports note gating doesn't always propagate reliably across agent-to-agent handoffs
User feedback	Copilot Studio "agent flows": built-in AI approval steps (Teams/Outlook approval cards) and request-for-information nodes that pause mid-run for structured human input
Gateways	Private networking for Foundry; Azure AI Content Safety Prompt Shields for jailbreak and injection detection; Microsoft Purview as the primary data-boundary control, covering DLP policies, sensitivity-label enforcement, unified audit logging, and an Insider Risk Management template specifically for risky AI usage

Common failure modes

- **EchoLeak (CVE-2025-32711, CVSS 9.3).** A zero-click prompt-injection vulnerability in Microsoft 365 Copilot, phrased to evade cross-prompt-injection classifiers and exploiting markdown link rendering to exfiltrate sensitive context with no user interaction. Microsoft patched it pre-disclosure and reports no confirmed customer compromise.
- **Copilot Studio SSRF (CVE-2024-38206).** Independent research found the `HttpRequestAction` connector could be abused to retrieve managed-identity credentials via instance metadata, and to reach an internal, multi-tenant Cosmos DB instance in Microsoft's own backend. Fixed after disclosure.

- **Approval bypass via injection.** Independent research (Zenity Labs, "AgentFlayer") describes prompt-injection techniques aimed specifically at circumventing the human-approval checkpoints described above, not only at unauthorized data access. A direct tension between the interception mechanisms Microsoft documents and their demonstrated bypassability in adversarial testing.

Credible control patterns

Treat topic and tool naming, not just instruction text, as a security-relevant design surface in Copilot Studio, since the planner weighs it more heavily than prose instructions. Wrap irreversible actions in the deterministic layer of the three-layer control model rather than exposing them directly to the generative orchestrator. In Foundry, apply `always_require` approval to any tool with real-world consequence by default rather than per-incident, given the documented reliability gap across agent-to-agent handoffs. Use Entra Agent ID's delegated-versus-application distinction deliberately: an agent acting under a user's own permissions carries a fundamentally different risk profile than one acting under its own standing RBAC role, and the choice should be explicit, not incidental to how the agent happened to get built.

4.4 OpenAI (Agents SDK)

How it works

OpenAI's current agent surface is the Responses API plus the Agents SDK (the Assistants API sunsets August 26, 2026). Agents are defined with static or dynamic `instructions`, and OpenAI has published a formal **instruction hierarchy** in its Model Spec: the model is trained to prioritize developer and system instructions over user input, and user input over third-party or tool content. A design-level attempt to make the model itself part of the control system, not just the harness around it.

Control points across the lifecycle

Stage	What's exposed
Design-time / system prompts	Static or dynamic `instructions` field; org-wide behavior baseline set by OpenAI's published Model Spec and instruction hierarchy
User input	`input`/`messages` arrays via `Runner.run()`; **input guardrails** as the first checkpoint, runnable in blocking mode so the agent never executes if tripped
Memory / context	Four conversation-memory strategies (`history`, `session`, `conversationId`, `previousResponseId`); a separate `RunContextWrapper` for app-local context explicitly never sent to the model

Stage	What's exposed
Tool calls	Standard function calling with `call_id`-keyed results; `ToolContext` exposes tool name, call id, and arguments to executing code; `Agent.as_tool()` for agent-as-tool orchestration
Identity / authorization	Org/project RBAC (Owner, Reader, Member, Viewer, plus 30+ granular custom permissions), project-scoped API keys, service accounts. This is IAM for developers accessing the API, not a runtime permission model for what a running agent may access; that's left to the developer's own tool implementations
Runtime decisions	`on_tool_start`/`on_llm_start` lifecycle hooks fire immediately before the respective action; **tool guardrails** (`ToolInputGuardrailTripwireTriggered`) can block at the individual tool boundary; `max_turns` caps runaway iteration
User feedback	Tools flagged `needsApproval: true` pause the run and emit a serializable `state` object; the app calls `state.approve()` or reject, and resumes the *same* run, supporting delayed or asynchronous human review
Gateways	Org/project-level rate limits (RPM/RPD/TPM/TPD); IP allowlisting (up to 50 IPs/CIDRs, Org Owner-gated); no documented outbound allowlist for what an agent's own tool calls may reach at the SDK level. That control exists at the product level (ChatGPT Atlas, for example), not as a generic developer-facing API gateway

Common failure modes

- **Prompt injection, self-described as possibly unsolvable for browser and computer-use agents.** OpenAI has stated this directly rather than treating it as a closed problem, and published a concrete internal red-team case: a malicious email with hidden instructions caused an agent asked to draft an out-of-office reply to instead send a resignation letter to the user's CEO.
- **Instruction hierarchy is a mitigation, not a guarantee.** The hierarchy shapes how the model weighs conflicting instructions but doesn't structurally prevent a sufficiently crafted injection from being interpreted as developer-level intent. It reduces the attack surface rather than closing it.
- **No default outbound gateway at the SDK level.** Unlike Anthropic's sandboxed network egress allowlisting, OpenAI's documented boundary controls (rate limits, IP allowlisting) govern access *to* the API, not what a running agent's own tool calls can reach outward. That gap is filled at the product level for OpenAI's own agent products, but is a build-it-yourself concern for developers using the SDK directly.

Credible control patterns

Run input guardrails in blocking mode, not parallel, for any agent handling untrusted input. Parallel mode trades safety for latency by letting the agent start before the guardrail resolves. Apply tool guardrails at the individual tool boundary rather than relying on agent-level instructions to constrain behavior, since guardrails aren't automatically inherited through handoff chains and need to be attached deliberately at each step. Treat `needsApproval` as the default for any tool with real-world consequence, and use its support for delayed or asynchronous review deliberately for actions that warrant more than a synchronous rubber stamp. Because there's no SDK-level outbound gateway, build one at the tool-implementation layer rather than assuming the platform provides it.

4.5 Custom / In-House

How it works

A custom harness, built on LangGraph, CrewAI, a raw model API loop, or something proprietary, has no vendor-supplied ceiling and no vendor-supplied floor. Every control point described in Section 2's lifecycle is *possible* to build. None of them ship by default. This is the structural difference from the four platforms above: their profiles describe what's available. This one describes what has to be decided.

Control points across the lifecycle

There's no fixed table here, because the answer for any given deployment is whatever the team building it chose to implement. What's consistent is the shape of the decision: for each of the eight lifecycle stages, a team building custom has to decide whether to instrument it, and with what. In practice, most custom harnesses inherit whatever their underlying model API provides at the tool-call and design-time stages, since most are built on top of Anthropic, OpenAI, or another model API's own tool-use loop, and build everything else (memory management, identity, runtime interception, human review) themselves, from scratch, with no vendor reference implementation to start from.

Common failure modes

- **Absence, not misconfiguration.** The dominant failure mode in custom builds isn't a control that was set up wrong. It's a control that was never built, because nothing in the framework prompted the team to consider it. A vendor platform's documentation is itself a checklist; a custom harness has none.
- **Framework defaults mistaken for security defaults.** Popular orchestration frameworks are built for capability and developer velocity, not security posture. A team that adopts a framework's default configuration is adopting its authors' priorities, which were very likely not what a security team should be able to intercept.

- **No shared incident record.** Because custom harnesses are, by definition, not shared infrastructure, there’s no equivalent of the CVE or incident record available for the four platforms above. Risk is discovered per-deployment, usually after the fact, rather than surfaced by a research community watching one common target.

Credible control patterns

Build the lifecycle in Section 2 into the design process itself: for each stage, make an explicit decision about whether it needs a control point, rather than defaulting to "we’ll add it if something goes wrong." Where the harness is built on top of a vendor model API, inherit that API’s own tool-call and design-time controls rather than reimplementing them, and concentrate custom engineering effort on the stages the API doesn’t cover: identity, runtime interception, memory governance. Treat the absence of a shared incident record as a reason for more scrutiny, not less. The lack of public CVEs against a custom harness reflects the size of the audience finding them, not the absence of the problem.

5. The Pattern Across Platforms

Read across the four platform profiles, three things hold consistently.

First, every platform has built real interception mechanisms. This field guide is not an argument that vendors have ignored the problem. Anthropic's hook system, OpenAI's guardrails and lifecycle hooks, Bedrock's Return of Control, and Microsoft's custom triggers and approval gating are all genuine, documented, usable controls. The gap this field guide surfaces isn't an absence of capability. It's that these controls are, in every case, opt-in and per-point: a team has to know a given hook exists, decide to configure it, and do so for each tool, action, or topic individually. None of the four ships a default posture of reviewing everything before it happens. That default has to be built by whoever is deploying the agent.

Second, the independent security research cited across these profiles converges on the same structural finding regardless of platform: the interception and approval mechanisms that exist can themselves be bypassed by a sufficiently crafted prompt injection. This showed up as social engineering of a Bedrock supervisor agent, as approval-bypass techniques against Copilot Studio, and as OpenAI's own acknowledgment that prompt injection against browser and computer-use agents may not be fully solvable. This isn't a criticism specific to any one vendor. It's evidence that a control point living entirely inside the model's own context, a hook that fires based on what the model decided to do, is only as trustworthy as the context that produced that decision. A hook the model can be talked out of triggering offers a weaker guarantee than one enforced independent of the model's own reasoning.

Third, identity and gateway controls remain the most mature part of every platform's stack, and decision-point control remains the newest. Entra Agent ID, Bedrock's shift from Classic's single execution role to AgentCore's per-workload identity, and Anthropic's sandboxed egress allowlisting are all substantially more developed than the runtime-interception layer sitting alongside them. That's a reasonable sequencing for the industry to have followed: identity and network boundaries are well-understood problems from pre-agent security, ported over. Decision-point control inside the agent's own execution path is a genuinely new problem, and it shows in how recent, how opt-in, and how inconsistent from vendor to vendor the tooling for it still is.

> **Where Geordie fits.** Geordie's own position is that this third gap, decision-point control operating at the moment an agent is choosing what to do next, applied through the agent's own context and configuration rather than an external gateway, is where the next phase of agentic governance concentrates. It's the one category present at every lifecycle stage, and it's the layer every platform in this field guide has left the least developed and the most opt-in. Geordie instruments the harness directly across cloud, code, and endpoint, with browser coverage coming, shaping agent behavior through

> deterministic, context-specific controls rather than a proxy sitting outside the agent's own execution path. This isn't a claim that the platform-native controls mapped in this field guide are wrong or unnecessary. Identity, gateways, and tool-calling boundaries each cover ground decision-point control doesn't reach on its own, and a mature posture draws on more than one. It's a claim about where the coverage gap sits today, the same gap the independent research cited throughout this field guide keeps finding, platform after platform, and the reason Geordie treats agentic governance as infrastructure for the operating model, not a bolt-on control.

Appendix

A. Methodology

Each platform profile was built from official platform documentation as the primary source, cross-checked against independent security research (Unit 42, Cloud Security Alliance, Tenable, Zenity Labs, Check Point Research, and others as cited) for failure modes and incidents. We prioritized primary vendor docs over third-party summaries wherever possible, and flagged any claim sourced from a community forum, unconfirmed report, or preview/beta feature as lower-confidence within the profile itself.

The matrix scoring in Section 3 is qualitative, not a weighted formula. Interception depth and decision-point granularity were assessed by mapping each platform's documented controls against the eight-stage lifecycle in Section 2 and judging how many stages have a genuine third-party interception point versus none. This is a judgment call, not a benchmark run against live systems, and we'd rather be transparent about that than present false precision. Where a platform's placement in the matrix sits close to another's, or where Classic and next-generation offerings (Bedrock Agents Classic versus AgentCore) diverge meaningfully, we've said so explicitly rather than collapsing to a single point.

This field guide does not cover pricing, ease of implementation, or overall platform quality. It covers one question: where can a third party intervene in what an agent is about to do, and how.

B. Sources

Anthropic

- Modifying system prompts: https://code.claude.com/docs/en/agent-sdk/modifying-system-prompts
- Hooks reference: https://code.claude.com/docs/en/hooks
- Permissions: https://code.claude.com/docs/en/agent-sdk/permissions
- Sandboxing: https://code.claude.com/docs/en/sandboxing
- MCP reference: https://code.claude.com/docs/en/mcp
- Tool use overview: https://platform.claude.com/docs/en/agents-and-tools/tool-use/overview
- Context editing: https://platform.claude.com/docs/en/build-with-claude/context-editing
- Memory tool: https://platform.claude.com/docs/en/agents-and-tools/tool-use/memory-tool
- Rate limits: https://platform.claude.com/docs/en/api/rate-limits
- Anthropic, "Disrupting the first reported AI-orchestrated cyber espionage campaign": https://www.anthropic.com/news/disrupting-AI-espionage

- Check Point Research, "Caught in the Hook" (CVE-2025-59536 / CVE-2026-21852): https://research.checkpoint.com/2026/rce-and-api-token-exfiltration-through-claude-code-project-files-cve-2025-59536/

Amazon Bedrock

- Bedrock Agents Classic maintenance mode: https://docs.aws.amazon.com/bedrock/latest/userguide/agents-classic-maintenance-mode.html
- How Amazon Bedrock Agents works: https://docs.aws.amazon.com/bedrock/latest/userguide/agents-how.html
- Return control to the agent developer: https://docs.aws.amazon.com/bedrock/latest/userguide/agents-returncontrol.html
- Retain conversational context across sessions: https://docs.aws.amazon.com/bedrock/latest/userguide/agents-memory.html
- Identity and access management for Amazon Bedrock: https://docs.aws.amazon.com/bedrock/latest/userguide/security-iam.html
- Identity and credential management for AgentCore: https://docs.aws.amazon.com/bedrock-agentcore/latest/devguide/identity.html
- Implement human-in-the-loop confirmation with Amazon Bedrock Agents: https://aws.amazon.com/blogs/machine-learning/implement-human-in-the-loop-confirmation-with-amazon-bedrock-agents/
- AWS Well-Architected Agentic AI Lens: https://docs.aws.amazon.com/wellarchitected/latest/agentic-ai-lens/agentsec04-bp02.html
- Unit 42, "When an Attacker Meets a Group of Agents": https://unit42.paloaltonetworks.com/amazon-bedrock-multiagent-applications/
- Unit 42, "When AI Remembers Too Much": https://unit42.paloaltonetworks.com/indirect-prompt-injection-poisons-ai-longterm-memory/
- Cloud Security Alliance, Bedrock AgentCore attack surface research note: https://labs.cloudsecurityalliance.org/research/csa-research-note-bedrock-agentcore-enterprise-attack-surfac/

Microsoft

- Apply generative orchestration capabilities: https://learn.microsoft.com/en-us/microsoft-copilot-studio/guidance/generative-orchestration
- Extend your agent with MCP: https://learn.microsoft.com/en-us/microsoft-copilot-studio/agent-extend-action-mcp
- What is Microsoft Entra Agent ID?: https://learn.microsoft.com/en-us/entra/agent-id/what-is-microsoft-entra-agent-id
- Governing Agent Identities: https://learn.microsoft.com/en-us/entra/id-governance/agent-id-governance-overview
- Using function tools with human-in-the-loop approvals: https://learn.microsoft.com/en-us/agent-framework/agents/tools/tool-approval

- Use Purview with Copilot Studio: https://learn.microsoft.com/en-us/purview/ai-copilot-studio
- Microsoft Agent 365 overview: https://learn.microsoft.com/en-us/microsoft-agent-365/overview
- Microsoft Agent 365 GA announcement: https://www.microsoft.com/en-us/security/blog/2026/05/01/microsoft-agent-365-now-generally-available-expands-capabilities-and-integrations/
- Dark Reading, "Researchers Detail Zero-Click Copilot Exploit 'EchoLeak'": https://www.darkreading.com/application-security/researchers-detail-zero-click-copilot-exploit-echoleak
- Tenable, "SSRFing the Web with the Help of Copilot Studio": https://www.tenable.com/blog/ssrfing-the-web-with-the-help-of-copilot-studio
- Zenity Labs, "AgentFlayer" disclosure: https://www.prnewswire.com/news-releases/zenity-labs-exposes-widespread-agentflayer-vulnerabilities-allowing-silent-hijacking-of-major-enterprise-ai-agents-circumventing-human-oversight-302523580.html

OpenAI

- Agents SDK overview: https://developers.openai.com/api/docs/guides/agents
- Running agents: https://developers.openai.com/api/docs/guides/agents/running-agents
- Context management: https://openai.github.io/openai-agents-python/context/
- Function calling guide: https://developers.openai.com/api/docs/guides/function-calling
- Guardrails: https://openai.github.io/openai-agents-python/guardrails/
- Guardrails and human review: https://developers.openai.com/api/docs/guides/agents/guardrails-approvals
- Manage permissions (RBAC): https://developers.openai.com/api/docs/guides/rbac
- Rate limits: https://developers.openai.com/api/docs/guides/rate-limits
- IP allowlist: https://developers.openai.com/api/docs/guides/ip-allowlist
- OpenAI Model Spec (2025-12-18): https://model-spec.openai.com/2025-12-18.html
- OpenAI, "Understanding the risks of prompt injections": https://openai.com/index/prompt-injections/
- OpenAI, "Hardening ChatGPT Atlas against prompt injection": https://openai.com/index/hardening-atlas-against-prompt-injection/

Full agent-by-agent research transcripts, including additional lower-confidence and community-sourced findings noted but not cited above, are retained internally and available on request.

C. Glossary

Agent. for this field guide, an LLM reasoning core connected to at least one tool. A deliberately low bar, chosen to cover single-tool copilots and multi-agent architectures under one consistent definition.

Harness. the framework, SDK, or platform that gives an agent its structure: memory, tools, and execution path. Distinct from the underlying model.

Control point. any point in the agent lifecycle where a third party can observe, interrupt, or shape what the agent does next.

Boundary control. a control that operates at the edge of the agent's execution, where it crosses into identity/authorization or network/gateway territory that exists independent of the agent itself.

Decision-point control. a control that operates inside the agent's own execution loop, shaping what it does through the context it's given rather than intercepting it at a boundary after the fact.

Interception depth. how many of the eight lifecycle stages a platform's native tooling exposes to third-party visibility at all.

Decision-point granularity. of the stages a platform exposes, how many can a third party actually act on before the agent proceeds, not just observe afterward.

D. Changelog

V1, August 2026. Initial publication. Covers Anthropic, Amazon Bedrock, Microsoft Copilot Studio/Azure AI Foundry, OpenAI, and custom/in-house builds.