AVIATRIX

SECURITY ARCHITECTURE BRIEF

Contain Enterprise AI Agents

Powered by Aviatrix + Microsoft

Threat model, enforcement architecture, and compliance evidence for the Microsoft Agent Control Specification integration

EXECUTIVE SUMMARY

AI agents are the fastest-growing and least-governed workload class in the enterprise. Organizations run several agent frameworks at once – LangChain, AutoGen, CrewAI, Semantic Kernel, Microsoft Copilot Studio, and vendor-shipped copilots – across Kubernetes, serverless, and managed cloud agent services. Each agent reaches Model Context Protocol (MCP) servers, large language model (LLM) APIs, and internal endpoints. Agent governance frameworks define what an agent may reach in code, but cannot guarantee every agent process loads and honors that policy. A vendor-shipped or compromised agent reaches the same tools, unguarded.

Microsoft's Agent Control Specification (ACS) is an open, portable policy contract for agent behavior. The Aviatrix Containment Plugin for Microsoft ACS is the enforcement plane that specification needs: an open-source shim compiles an ACS policy file into Aviatrix Distributed Cloud Firewall (DCF) rules. Enforcement is Communication Governance at the network – outside the agent's trust boundary – so an agent that ignores its own policy still cannot reach an unauthorized tool. One policy file produces the same posture across every framework, cloud, and runtime.

The strategic outcome is a bounded Blast Radius for every agent, governed or not. This brief gives the security architect what is needed to approve the control before procurement: the full threat model, the kill chain and point of intervention, how enforcement works, what the control explicitly does not do, and the compliance evidence it produces. Deployment specifics live in the companion Technical Brief.

Threat Model

The threat is an AI agent reaching a tool endpoint it should not – to exfiltrate data, invoke an unauthorized capability, or pivot through an MCP server – whether because the agent was compromised, never integrated with governance, or simply misbehaved. The agent is the vehicle; the reachable tool endpoint is the target.

Threat property	Why it defeats framework-only governance
Policy lives inside the agent	Governance defined in the framework wrapper executes inside the agent's trust boundary. A compromised agent process can ignore its own policy file and call any reachable endpoint.
Unintegrated agents	Vendor-shipped copilots and third-party agents have no integration point with the governance framework. No policy is applied to them at all.

Legitimate-looking traffic	A tool call is an outbound connection that looks like every other tool call. When it is malicious, detection has no anomaly to find.
Autonomy and privilege	Agents make non-deterministic outbound calls with broad, inherited privilege. Compromise of the workload is compromise of its reach.

WHY THIS MATTERS

No agent framework can guarantee that every agent process in an estate loads and honors its policy contract. The foundational invariant – that no unguarded path to a tool endpoint exists – can only be enforced where every agent's traffic must pass: the network. Microsoft's ACS reference architecture uses a service mesh for transport only; it carries the policy file to the sidecar and stops there. It does not compile policy into enforcement, does not apply pattern-guard inspection, and covers only the Kubernetes mesh. Enforcement at the wire is the control the specification itself cannot provide.

Attack Scenario and Kill Chain

The kill chain below traces an agent reaching an unauthorized tool endpoint and identifies where containment intervenes.

Kill Chain Stage	What Happens	Control
Agent compromise or gap	An agent is compromised through its trust chain, or a vendor-shipped agent runs with no governance integration.	Out of scope – containment assumes the agent may be compromised or ungoverned.
Policy bypass	The agent ignores or never loaded its ACS policy and prepares a call to an unauthorized MCP server or LLM API.	Not prevented at the agent – enforcement does not depend on agent cooperation.
Tool-call attempt	The agent opens an outbound connection to the unauthorized endpoint.	DENIED. Default-deny DCF policy blocks any destination not in the agent's permitted WebGroups, and logs it.
Sensitive-data egress	A permitted call carries sensitive content in its payload.	Input/output pattern guards apply inline intrusion prevention system (IPS) inspection today; semantic detection and redaction arrive with the Guardrail Profile (late summer 2026).
Audit trail	Responders need to know what was attempted and whether it was stopped.	DCF flow logs and audit events record every rule hit in CoPilot, correlated with ACS policy events by session identifier.

POINT OF INTERVENTION

Aviatrix intervenes at the tool-call attempt – the one stage where the agent must traverse the network to succeed. The agent workload is scoped by a SmartGroup; the endpoints it may reach are defined as WebGroups; DCF evaluates every outbound connection against a default-deny posture before it leaves the environment. A call to an unauthorized MCP server or LLM API hits a deny rule whether or not the agent has any awareness of ACS. The agent runs; the unauthorized reach does not.

Enforcement Architecture

Enforcement is Communication Governance at the network layer, built entirely on the Aviatrix Distributed Cloud Firewall. An open-source shim compiles the ACS policy file into DCF objects; no agent code changes are required.

Compilation, not transport

A command-line shim reads the ACS guardrails file and emits DCF objects: the resource catalog becomes WebGroups, the agent identity becomes a source SmartGroup, and block/allow policies become DCF rules inside a default-deny posture. Input and output pattern guards become inline IPS rules. The shim can apply the generated policy through the Aviatrix REST Application Programming Interface (API) or the Terraform provider, so policy ships as infrastructure-as-code.

Enforcement outside the agent's trust boundary

DCF evaluates every outbound connection at the gateway. There is no sidecar requirement and no mesh dependency. Because enforcement is a property of the network path rather than a process on the workload, a compromised agent cannot disable it. This is what separates containment from in-framework filtering: the wall exists before the breach and holds regardless of the agent's behavior.

Framework- and runtime-agnostic coverage

Any agent workload that generates network traffic and runs in an environment served by an Aviatrix gateway is covered equally – across agent frameworks, vendor-shipped agents, Kubernetes on any cloud, virtual machines, serverless functions, managed cloud agent platforms, and on-premises workloads. An enterprise running multiple frameworks across multiple clouds gets one consistent policy and one unified audit trail.

WHY THIS IS DIFFERENT FROM A SERVICE-MESH REFERENCE

A service mesh in the ACS reference architecture carries the policy file and stops there – no compilation into enforcement, no pattern-guard inspection, Kubernetes-only, and no external authorization integration. The Containment Plugin does the enforcement work: it compiles the specification into DCF policy, applies IPS guards, and covers virtual machines, serverless, and multiple clouds. Enforcement at the wire is the category difference.

Architecture reference: *ACS policy file → shim compiles → DCF SmartGroups (agent identity) + WebGroups (permitted MCP/LLM endpoints) + allow/deny rules + inline IPS guards → gateway enforces default-deny on every agent's outbound traffic → flow logs and audit events to CoPilot. Deployment topology is detailed in the Technical Brief.*

Architectural Boundaries

Containment governs reachability – what an agent can reach on the network. Stating what it does not do is part of approving it.

Out of scope	What governs it instead / status
MCP-over-stdio traffic	Agents talking to a local MCP server over standard input/output generate no network traffic and are invisible to DCF. Recommendation: deprecate stdio MCP in governed estates.
Stateful ACS predicates	Sensitivity ratchets, session variables, conditional rules, and human-approval gates require stateful evaluation – delivered by the Guardrail Profile feature (coming soon). The coverage report lists these as pending.
Semantic content inspection	Deep payload understanding, advanced PII detection, and redaction arrive with the Guardrail Profile. Traditional IPS pattern guards apply today.
Tool-to-host binding	Agent FQDNs are not in the ACS spec. The shim requires an operator-supplied tool-to-host registry, or gateway inspection, to bind tool names to destinations for WebGroup construction.

Compliance Evidence

For SOC 2, PCI-DSS, HIPAA, and FedRAMP environments, auditors require architectural proof of enforcement – not a policy document. This architecture produces evidence as a by-product of how it runs.

Evidence artifact	What it proves
Default-deny policy definition	Agent outbound traffic is restricted to explicitly approved tool endpoints (PCI-DSS outbound restriction; FedRAMP least-functionality).
Per-rule flow logs and audit events	Every allowed and denied agent connection is logged in CoPilot with source identity and destination (SOC 2 audit logging; FedRAMP continuous monitoring).
Agent SmartGroup segmentation	Agents handling sensitive data are network-segmented from the rest of the estate (HIPAA segmentation for protected health information).
Policy-as-code change history	The shim emits DCF objects managed via Terraform or the REST API, reviewed in the same pull request as deployment manifests – an auditable approval trail (SOC 2 change management; NIST Secure Software Development Framework).
Coverage report	A by-rule statement of what is enforced at the network today versus pending the Guardrail Profile – evidence that the control's scope is documented, not assumed.

The proof of enforcement is the architecture itself – not a statement about the architecture.

Deployment Paths

The architecture deploys on a standard Aviatrix Distributed Cloud Firewall footprint, with no agent code changes and no sidecar or mesh dependency. Enforcement attaches at the network boundary serving each agent runtime, so the same policy model covers every environment:

- Kubernetes (any cloud) – the cluster network attaches to an Aviatrix Spoke Gateway; pod egress is governed at the node subnet boundary.
- Virtual machines and serverless functions – agent workloads egress through the gateway serving their network.
- Managed cloud agent platforms and on-premises workloads – governed wherever an Aviatrix gateway sees their traffic.

The shim compiles the ACS policy file into Distributed Cloud Firewall objects and applies them through the REST Application Programming Interface (API) or the Terraform provider, so policy ships as infrastructure-as-code. New policy deploys in monitor mode first to baseline real agent traffic, then promotes to enforce, scoped to one agent group at a time and reversible without redeploying the gateway. Step-by-step deployment, prerequisites, and policy structure are in the companion Technical Brief.

Next Steps

Request a 30-minute architecture review

We walk through the enforcement model in your environment, map your current agent inventory and the tool endpoints they reach, and identify the paths you currently cannot see. Deployment specifics – insertion pattern, prerequisites, policy structure, and constraints – are in the companion Technical Brief, which ships with the deployment repository.

About Aviatrix

Aviatrix® is pioneering the Cloud Native Security Fabric – the architecture the Containment Era requires. The Cloud Native Security Fabric governs every workload communication path across every cloud, every VPC, every Kubernetes cluster, and every serverless function, from a single policy plane. One rule. Universal propagation. Enforced at the workload, not at a chokepoint. Trusted by more than 500 of the world's leading enterprises. For more information, visit aviatrix.ai.