AVIATRIX

SECURITY ARCHITECTURE BRIEF

Contain Enterprise MCP Infrastructure

Powered by Aviatrix + Obot

Threat model, enforcement architecture, and compliance evidence for security architecture review

EXECUTIVE SUMMARY

Aviatrix Validated Containment Architectures are lab-tested containment deployment blueprints for the AI platforms enterprises are actually running. They are ship-ready, policy-included, and validated before they arrive. This Validated Containment Architecture covers Enterprise MCP Infrastructure with Obot.

Every enterprise Model Context Protocol (MCP) server is an ungoverned egress path by default. When a dependency is compromised, it inherits the server's network permissions and can exfiltrate data to any destination. Kubernetes NetworkPolicy, service meshes, and vulnerability scanners cannot catch this. This architecture breaks that attack chain at the network layer.

Obot governs which MCP servers run; Aviatrix controls where they can reach. Enforcement is built into the deployment, not bolted on afterward. The result is per-server egress containment with full audit trail, enforced across Azure Kubernetes Service (AKS) and Amazon Elastic Kubernetes Service (EKS) without code changes or sidecars.

Threat Model

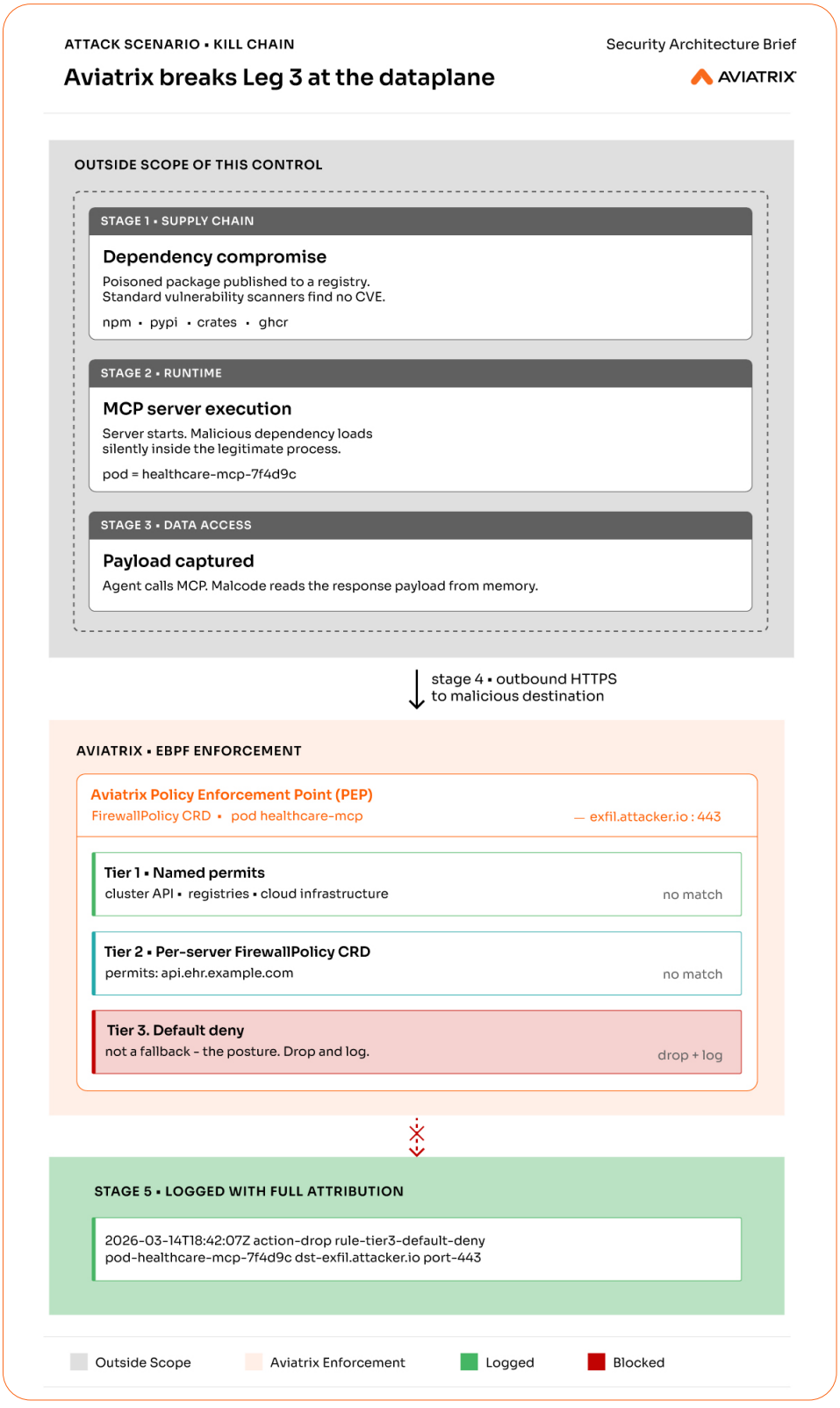
The Lethal Trifecta describes the three conditions that together create an exfiltration vector in agentic AI systems:

Lethal Trifecta	MCP Server Exposure
Leg 1: Private data access	MCP servers handle requests for data on behalf of AI agents. A healthcare MCP server may proxy patient records. A coding MCP server may hold API keys. Data access is the design intent.
Leg 2: Untrusted content exposure	MCP servers process user-supplied prompts, tool responses, and external data. Any of these can be adversarially crafted. Indirect prompt injection is an established and growing attack class.
Leg 3: External communication	By default, an MCP server can reach anything its host can reach. Most enterprises run more MCP servers than they track. Few have per-server egress controls. This is the leg that turns compromise into breach.

Legs 1 and 2 are intrinsic to MCP server design. This architecture breaks Leg 3 at the network layer.

WHY SUPPLY CHAIN ATTACKS MAKE LEG 3 THE PRIORITY

The March 2026 supply chain wave compromised Axios (70 million weekly downloads), LiteLLM, Trivy, KICS, and Telnyx. A poisoned dependency inside an MCP server inherits the server's egress permissions automatically. The server itself is not compromised, but the dependency is. Trust-based controls don't catch this. Domain-based enforcement at the network layer does, because it doesn't care whether the code running is legitimate or poisoned. It only cares where the traffic is going.



Attack Scenario and Kill Chain

An MCP server is permitted to send patient data to an Electronic Health Record (EHR) endpoint. A compromised npm dependency, a poisoned package, attempts to exfiltrate the same data to an attacker-controlled domain.

Kill Chain Stage	What Happens	Control
Dependency compromise	Poisoned package published to registry. Passes standard vulnerability scanning: no known CVE at time of install.	Outside scope of this control
MCP server execution	Server starts. Malicious dependency loads. Server has legitimate permission to access patient data.	Outside scope of this control
Data access	Agent calls MCP server. Server fulfills the request. Malicious code captures the response payload.	Outside scope of this control
Exfiltration attempt	Malicious code initiates outbound HTTPS to exfil.attacker.io on port 443.	Aviatrix blocks destination not in permit list
Audit trail	Blocked connection logged at the Policy Enforcement Point (PEP), the Aviatrix Spoke Gateway, with pod identity, domain, timestamp, and policy rule.	CoPilot audit trail, attributable to specific MCP server

POINT OF INTERVENTION

The server has permission for the action, not the destination. The FirewallPolicy Custom Resource Definition (CRD) for this server permits the EHR endpoint only. Aviatrix evaluates the destination at the eBPF dataplane and drops the connection before it leaves the VPC/VNet. Patient data does not leave the environment. The attempt is logged with full attribution.

REFERENCE ARCHITECTURE CONCEPTUAL

Two-layer enforcement

Solution Brief

AVIATRIX + obot

LAYER 1 ▪ OBOT ▪ GOVERNANCE

Which MCP servers run

Each MCP server's manifest declares the domains it may reach. Empty list = zero egress.

namespace: obot-mcp

mcp-server
healthcare-mcp
permits: api.ehr.example.com

mcp-server
code-mcp
permits: api.github.com

mcp-server
internal-mcp
permits: (none) · zero egress

every packet from every pod

LAYER 2 ▪ AVIATRIX ▪ ENFORCEMENT

Where they can reach

The Aviatrix gateway evaluates each server's permit list inline. Three tiers nothing falls through.

Aviatrix Policy Enforcement Point (PEP)
eBPF dataplane inline policy evaluation

Tier 1 ▪ Named permits
Cluster API ▪ container registries ▪ cloud infrastructure — allow

Tier 2 ▪ Per-server permits
Match by pod label ▪ allow declared domains only — deny/allow

Tier 3. Default deny
The posture, not a fallback. — deny + log

EXTERNAL DESTINATIONS

Cluster infrastructure
K8s API ▪ registries

Permitted domains
api.ehr.example.com
api.github.com

Cluster infrastructure
K8s API ▪ registries

Obot governance · Aviatrix enforcement · Permitted · Blocked + logged

Enforcement Architecture

Obot controls the server lifecycle. Aviatrix enforces the network path. Enforcement runs at two layers, and neither requires code changes, sidecars, or application team involvement.

Layer 1 – Obot: Governance of Which Servers Run

Obot deploys MCP servers into an isolated Kubernetes namespace via its controller. Each server is declared with an explicit list of domains it is permitted to reach. The controller reconciles a FirewallPolicy CRD for each server and triggers cleanup on deletion.

- Obot controls spawn, configuration, and teardown
- Egress domains are declared in the server manifest and versioned in git
- A server with an empty domain list has zero egress; containment is the default when deployed with the Aviatrix Terraform blueprint. Obot also supports an allow default egress policy per server. When set to allow, no FirewallPolicy CRD is created and egress is unrestricted. The reference architecture sets deny as the default. Verify this setting in every server manifest.
- Deleting a server automatically removes its network permits: no orphaned rules

Layer 2 – Aviatrix: Enforcement of Where Servers Reach

Traffic from each MCP server pod passes through the PEP (Aviatrix Spoke Gateway). The eBPF dataplane evaluates the server's FirewallPolicy CRD inline. Three-tier evaluation ensures nothing falls through:

Tier	Role
Tier 1: Named permits	Explicit permits for cluster infrastructure and cloud services. Evaluated first.
Tier 2: Per-server rules	Per-MCP-server FirewallPolicy CRD. Permits declared domains. Traffic not matching falls to Tier 3.
Tier 3: Default deny	Catches everything not explicitly permitted in Tier 1 or Tier 2. This is not a fallback; it is the posture.

WHY THIS IS DIFFERENT FROM KUBERNETES NETWORKPOLICY

NetworkPolicy operates on IP and port. It cannot distinguish two MCP servers in the same namespace for egress policy. Aviatrix FirewallPolicy CRDs operate on pod label and domain: each server gets its own egress permit list. Two servers in the same namespace can have completely different policies.

Architectural Boundaries

This control governs where MCP server pods can send traffic. It is one layer in a defense-in-depth posture, not a complete AI security solution.

Out of Scope	What Governs It Instead
Payload inspection	Domain-based, not content-based. Payload inspection requires a separate control.
Prompt-level guardrails	LLM input/output inspection is the function of AgentGuard Advanced Guardrails (HiddenLayer integration).
Shadow AI discovery	Discovering ungoverned AI workloads is the function of AgentGuard Shadow AI Discovery.
East-west traffic	This control governs external egress at the PEP. Internal cluster traffic requires separate Distributed Cloud Firewall (DCF) policy.

Compliance Evidence

For HIPAA, PCI-DSS, SOC 2, and ISO 27001 environments, auditors require architectural proof of enforcement, not a policy document.

Evidence Artifact	What It Proves
FirewallPolicy CRDs in git	Security policy in the same repo as the deployment manifest. Same PR. Same pipeline. Every change tracked, attributed, and reversible.
kubectl-inspectable policy	kubectl get firewallpolicy -n obot-mcp shows exactly which domains each server may reach at the moment of inspection. No interpretation required.
Kubernetes audit log	Every CRD creation, modification, and deletion tracked with timestamp and author.
PEP block logs	Every blocked attempt logged with pod identity, destination domain, policy rule, and timestamp. Exportable to SIEM.
Automatic permit cleanup	Deleting an MCP server removes its FirewallPolicy CRD automatically. No orphaned permits. Auditors can verify programmatically.

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

Deployment Paths

Three paths accommodate existing Aviatrix customers, net-new deployments, and teams adding enforcement to an existing Kubernetes cluster.

01 **Existing Aviatrix Cloud Native Security Fabric customers**

If Obot is being deployed to a cluster already attached to an Aviatrix spoke and onboarded in CoPilot, no new gateway is required. Install Obot >= 0.21.0 via Helm with the aviatrix-network-policy-controller enabled. The controller reconciles MCPNetworkPolicy objects into FirewallPolicy CRDs on the existing Aviatrix controller. Same console, same control plane, same policy model as every other workload in the fabric.

02 **Net-new Aviatrix deployments**

Deploy a Controller and CoPilot first (AWS/Azure Marketplace, or Aviatrix Cloud Fabric), and onboard the target cloud account. Then deploy from the obot-mcp-egress-azure (AKS) or obot-mcp-egress-aws (EKS) Terraform blueprint. One terraform apply provisions the Kubernetes cluster, Aviatrix Spoke Gateway, Obot platform (>= 0.21.0), and all DCF enforcement. No transit required: the Spoke Gateway operates standalone. Deployment time: 15-30 minutes. Destroy: one command, zero orphaned resources.

03 **Existing Kubernetes clusters without Aviatrix**

For teams already running EKS or AKS without Aviatrix. Deploy a Controller and CoPilot first and onboard the target cloud account. Then deploy an Aviatrix Spoke Gateway into the existing VPC or VNet, route the cluster node subnet default egress through it, and onboard the cluster in CoPilot. Install the k8s-firewall Helm chart to provision the Aviatrix CRDs, then deploy Obot >= 0.21.0 with the network-policy-controller enabled. The Terraform blueprint is a reference for the Aviatrix and Helm resource configuration. AKS prerequisite: ip-masq-agent must be configured with nonMasqueradeCIDRs: 0.0.0.0/0 so that pod source IPs reach the gateway unmasqueraded. Without this, label-based CRD enforcement silently fails. See the Technical Brief for full prerequisites.

Next Steps

The Validated Containment Architecture for Enterprise MCP Infrastructure with Obot is available today. The Terraform blueprint, scenario user interface, and deployment guide ship together. Aviatrix Controller 8.2 or later is required.

Request a 30-minute architecture review

We walk through the enforcement model in your environment, map your current MCP server inventory, and identify egress paths that may lack visibility today.

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.