The Semantic Proxy Pattern

A defense-in-depth architecture for enterprise AI agent authorization

Attribit-ID · April 2026

Three layers. Independent threat vectors. Compounding effectiveness.

- A mandatory semantic proxy evaluates every agent action against allowlist policy — out-of-band, invisible to the agent, fail-closed by design.
- Subnet isolation makes proxy traversal topologically mandatory — no agent packet leaves without proxy authorization.
- Per-agent cryptographic identity enables instance-level audit, role-specific policy, and granular revocation without fleet disruption.

The Threat: Prompt Injection as RCE

Prompt injection — embedding malicious instructions in data an agent processes — has no direct analog in traditional vulnerability taxonomies. Its functional effects most closely resemble remote code execution.

When an agent with tool access processes a malicious instruction, it executes with the agent's full permissions. The audit trail records the action as legitimate automation. Standard endpoint detection does not catch it.

In 2025–2026: EchoLeak (CVE-2025-32711), GitHub Copilot RCE (CVSS 9.6), Cursor IDE (CVSS 9.8). Google forecasts rising targeted attacks throughout 2026 against organizations that have connected agents to sensitive systems without real-time action monitoring.

The Identity Gap

Most deployments share API keys across agent instances. Four consequences:

No attribution. Forensics stops at "the agent fleet made these requests" — not which instance, task, or moment.

No granular revocation. Revoking a compromised agent rotates credentials shared across hundreds of instances.

Impersonation risk. Without per-agent cryptographic identity, Agent A can make requests appearing to originate from Agent B.

Static over-privilege. An agent that runs for five minutes holds credentials valid for hours after its task completes.

Why Traditional Controls Fail

Network security is syntactic. Firewalls operate on IPs, ports, and packet signatures. They cannot evaluate whether an agent *should* be making a valid HTTPS request to an approved API. Exfiltration through an approved endpoint is invisible.

RBAC is static. Role-based access cannot evaluate whether a specific action within a permitted role is appropriate given current context. An agent assigned "email-sender" can send *this* email — but should it?

Guardrails in the primary model are probabilistic. Research on 300,000 adversarial prompts: basic system prompt defenses allow 7% of attacks through. More fundamentally, placing the security control *inside the entity being attacked* is architecturally unsound.

Layer 1: The Semantic Proxy

Every outbound agent request passes through a proxy before reaching the network. The proxy submits each request — with agent identity, policy context, and session history — to a separate policy evaluation LLM. The monitor evaluates whether the action conforms to allowlist policy. Pass: forwarded. Fail: blocked and logged.

The critical property: out-of-band positioning. The primary agent does not know the proxy exists. It cannot reason about, circumvent, or negotiate with a control it cannot perceive. A prompt injection attack that compromises the model's instruction-following cannot disable monitoring it has no visibility into.

Allowlist, not denylist. Allowlist fails closed — unknown actions are blocked. Denylist fails open — every policy gap is a potential exploit path.

Layer 2: Subnet Isolation

Without topology enforcement, the proxy is advisory. Subnet isolation makes it mandatory.

All agent execution environments are placed in a dedicated subnet. Default egress: deny all. Single permitted egress: outbound to the proxy on defined ports. Enforced at the hypervisor or network device layer — outside the control of any agent instance.

Result: A compromised agent cannot reach the corporate network, internal databases, or external systems except through the proxy. Blast radius is bounded by what the proxy permits — which is bounded by the allowlist. Lateral movement becomes topologically impossible before any policy evaluation occurs.

Layer 3: Per-Agent Identity

SPIFFE/SPIRE issues cryptographic identities to workloads based on runtime attestation. An agent container receives a unique identity certificate valid only for its runtime. When it terminates, the identity expires. No static credential to protect.

Short-lived JWTs scoped to the specific task and valid only for expected task duration (minutes). The proxy validates the JWT on every request — not just at session establishment.

What this enables: Per-instance audit trails. Role-specific allowlist policies per agent. Granular revocation — compromise one instance, revoke it, fleet continues. Differentiated traffic paths with controls appropriate to agent vs. human populations.

Threat Analysis in Brief

Attack	Defense mechanism	Residual risk
Prompt injection	Proxy blocks action before network; agent's model-level compromise doesn't propagate	Subtle policy-conforming exfiltration
Lateral movement	Subnet egress rules — topological, not policy	None at network layer
Credential theft	Tokens are session-scoped, expire in minutes, real credentials never leave proxy	Detection latency
Agent impersonation	mTLS/JWT auth — Agent A cannot present Agent B's certificate	Key compromise

Implementation Roadmap

Phase 1 — Proxy deployment (Weeks 1–6): single workload, monitoring-only mode for 2 weeks to characterize normal action profile, then blocking mode.

Phase 2 — Subnet isolation (Weeks 4–10, parallel): dedicated agent VPC/namespace, deny-all egress except proxy, validate with penetration test.

Phase 3 — Per-agent identity (Weeks 8–16): SPIFFE/SPIRE or OAuth 2.1 authorization server, per-instance audit trail, granular revocation testing.

Phase 4 — Fleet expansion: each new agent type requires allowlist policy definition, subnet assignment, identity provisioning. Establish policy review cadence.

Five questions this answers

- Why do existing network security controls fail against semantic agent threats?
- How does an out-of-band proxy differ from guardrails embedded in the primary model?
- What implementation components are needed for each of the three architectural layers?
- How does this architecture perform against prompt injection, lateral movement, and impersonation?
- What does a phased implementation roadmap look like for enterprises?

The proxy doesn't make agents trustworthy. It makes their actions verifiable and their failures containable.

That is a tractable engineering problem. The tools exist today.

attribit-id.com/writing/semantic-proxy-pattern