Treat Your AI Agents Like Untrusted Code

Why the proxy architecture — semantic enforcement, subnet isolation, per-agent identity — is the engineering answer to agent security

Attribit-ID · April 2026

The moment your agent can send an email, it can forward confidential data

Critical CVEs in Microsoft Copilot (CVSS 9.3), GitHub Copilot (CVSS 9.6), and Cursor IDE (CVSS 9.8) all exploit the same weakness: AI agents operating with ambient trust and no external enforcement boundary. This is not theoretical. This is production.

The Problem Is Architectural, Not Behavioral

Traditional security controls are *syntactic* — IPs, ports, packet signatures. None of them have any idea what your recruiting agent intends to do when it initiates an HTTP connection to an external server.

You cannot solve a semantic problem with syntactic tools.

Agents have no inherent identity in most current deployments. You know they have an API key. You don't know which specific instance is using it, whether its runtime has been tampered with via prompt injection, or how to revoke access for a single agent without disrupting the entire fleet.

Prompt Injection = Remote Code Execution

When an agent has tool access, a malicious instruction hidden in a PDF, email, or document doesn't just corrupt the model's output.

It executes with the agent's full permissions. Against real systems. With an audit trail that records the action as legitimate automation.

Google's threat forecasting identifies targeted prompt injection against enterprise AI as one of the fastest-growing attack vectors of 2026.

What Brex Built — and Why It Works

Pedro Franceschi's "crab trap": an HTTP proxy that intercepts all agent traffic and routes it through a separate LLM that screens each action against defined policy, then blocks at the network layer — *without the primary agent knowing the control exists*.

The out-of-band positioning is what gives this architecture its teeth. An agent cannot reason its way around a control it has no visibility into. A prompt injection attack that compromises the primary model cannot disable monitoring it cannot perceive.

You don't need to trust the agent. You need to trust the proxy and the topology.

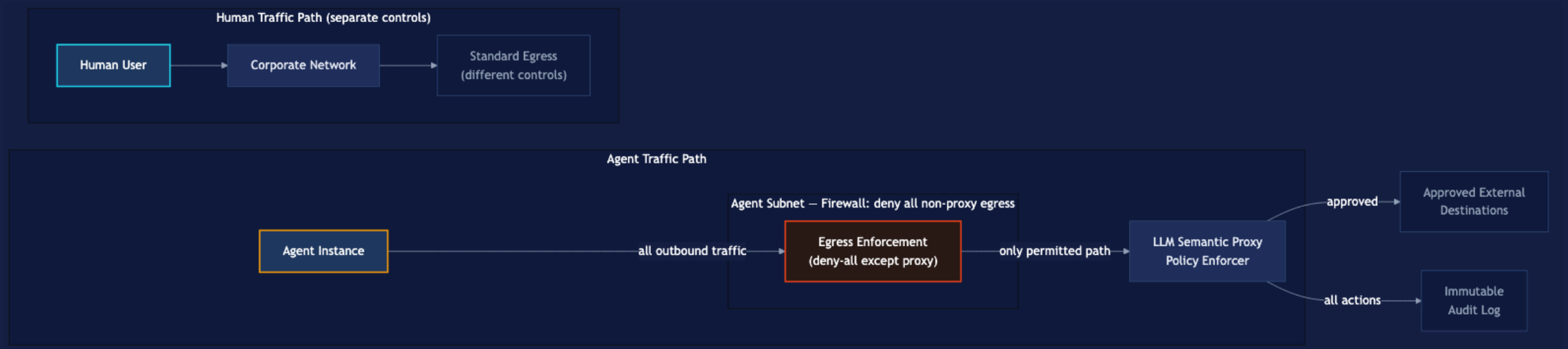
Three Layers That Must Compound

Layer 1 — Semantic proxy: Every action evaluated against allowlist policy by a second AI system before it reaches the network. Allowlist fails closed. Denylist fails open. This distinction is not administrative preference — it determines whether novel attacks succeed by default.

Layer 2 — Subnet isolation: All agent egress forced through the proxy by firewall rules enforced at the hypervisor layer. A compromised agent cannot reach corporate infrastructure without proxy authorization. Lateral movement is bounded by topology, not policy.

Layer 3 — Per-agent identity: mTLS certificates or short-lived JWTs scoped to individual task instances. Unlocks role-specific policy, granular revocation, forensic attribution, and impersonation detection simultaneously.

The Combined Architecture



The Compounding Effect

Research on 300,000 adversarial prompts:

Basic system prompt defenses alone — **7% of attacks through.**

Add content inspection — **0.2% of attacks through.**

Add prompt injection detection — **0.003% of attacks through.**

A **2,300-fold improvement** from layered defense. Three independent layers, each with different failure modes. All three must fail simultaneously for a breach to succeed.

Three questions this answers

- Why can't traditional firewalls and RBAC protect against prompt injection in AI agents?
- What is the semantic proxy pattern and why does out-of-band positioning matter?
- How do three independent security layers — proxy, subnet, identity — compound to contain agent compromise?

You don't have to solve AI alignment. You have to build a boundary you can trust around an agent you can't fully predict.

That boundary is available to build today.

attribit-id.com/writing/treat-agents-like-untrusted-code