MAYA

Not Your Daddy's Firewall

Why the perimeter still matters, and why it was never built to govern your agents

A Maya founder whitepaper · By Daljeet Singh, Founder · Mayagentic Inc.

Your firewall isn't obsolete. It's just facing the wrong direction.

For thirty years the perimeter did an honest job: it stood at the edge and watched what came *in*. That job hasn't gone away, and Maya has no intention of taking it. But your agents didn't arrive through the front door. They're already inside, talking to each other, reaching out to the world, acting on their own, and the instrument you bought to watch the door was never designed to see them.

This is a paper about that gap: what the perimeter was built for, what it can't do, and the different shape of control that agents actually require.

1. In praise of the perimeter

Let's be clear-eyed about what a web application firewall, a next-gen firewall, or an edge proxy is *for*. It defends the north-south boundary: the untrusted outside trying to get to your trusted inside. SQL injection, cross-site scripting, credential stuffing, volumetric DDoS, known-bad signatures, bot traffic. This is inbound threat, moving from north to south, and the perimeter is genuinely good at stopping it.

None of that changes because you deployed agents. If anything, an internet full of autonomous attackers makes the edge *more* important, not less. So this paper is not an argument that the firewall is dead. It's an argument that the firewall is an **inbound-facing instrument**, and the risk introduced by agents doesn't come from the direction it's facing.

Maya sits behind the perimeter, not on top of it. Different vector, different job.

2. The traffic the perimeter never watched

Agents changed the topology of risk. The action is no longer at the edge; it's *inside*, and it moves along two vectors the perimeter barely sees and cannot attribute:

East-west (E-W): agent-to-agent. Agents delegate to each other, orchestrate each other, and call each other's tools. When a prompt-injected agent's first move is to talk to an agent it was never supposed to reach, that conversation never crosses the perimeter at all. **South-north (S-N):** agent-initiated egress. An agent inside your environment reaches *out*: to a model provider, an API, a tool server, the open internet. The perimeter is built to inspect what comes in; it was never designed to reason about what your own workloads decide to originate.

The firewall guards the front door. Agents work the hallways and slip out the side. And here's the structural reason it can't follow them: the perimeter identifies traffic by **location**. IP, port, five-tuple. Agents arrive in fleets, share hosts and IP addresses, appear and vanish in seconds, and talk over TLS the network can't read. The moment your workloads became autonomous and numerous, the network lost the ability to say *which agent* did anything. Every control built on network location inherits that blindness.

3. The paradigm gap

Suppose the firewall *could* see the east-west and south-north traffic. It still couldn't govern it, because its entire model is the wrong shape.

A firewall thinks in **match-criteria → action**: if a packet matches this five-tuple or that signature, then allow, deny, or log. It's a lookup table of static conditions over packets. That model works beautifully for a world of stable hosts and enumerable rules.

Now try to write a match-criteria rule for: "*the billing agent may delegate to the ledger agent, but may not reach any external model provider except the two on our approved register, and must never accept an inbound connection*." You can't, not durably. The agents are ephemeral, they collapse onto shared IPs, and the thing you're trying to express isn't a property of a packet. It's a property of an **agent's intent**.

That's the paradigm break. The agentic view doesn't ask "does this packet match a bad pattern?" It asks "is this agent doing what it was declared to do?" You stop enumerating forbidden packets and start describing intended behavior: per agent, by identity.

4. Deviation: the delta between observed and declared

This is the primitive the rest of the paper is built on.

The operator **declares** a small set of facts about each agent: what it may reach, who it may talk to, which providers it may use, where it may run, how long it should live. Maya **observes** what each agent actually does, derived from the wire, without an SDK, a sidecar, or a single byte of payload. The gap between the two is a **deviation**: a typed event, emitted the moment observed contradicts declared.

Deviation reframes security as a control problem, not a hunting problem. You're no longer chasing an unbounded space of bad packets; you're measuring drift from a stated baseline. In control-theory terms, the declaration *is* the setpoint, and a deviation is the delta from it. **Compliance and governance want that delta at zero**: observed behavior held at what was declared.

The stakes are not hypothetical. Recent industry surveys put the share of enterprises with unknown AI agents running in their infrastructure at over 80%, while only a small fraction have real, real-time visibility into how those agents touch data. Machine identities now outnumber humans by a wide margin, and most organizations admit they lack identity controls for them. You cannot hold a fleet at setpoint if you cannot see the fleet.

Two honest boundaries, because a serious reader will look for them. First, Maya's declarations are **operator assertions, not attestations**: Maya catches an agent that contradicts its claim; an agent that declares broadly and stays within that broad claim isn't caught by contradiction, so over-broad declaration is itself scored and surfaced as a posture

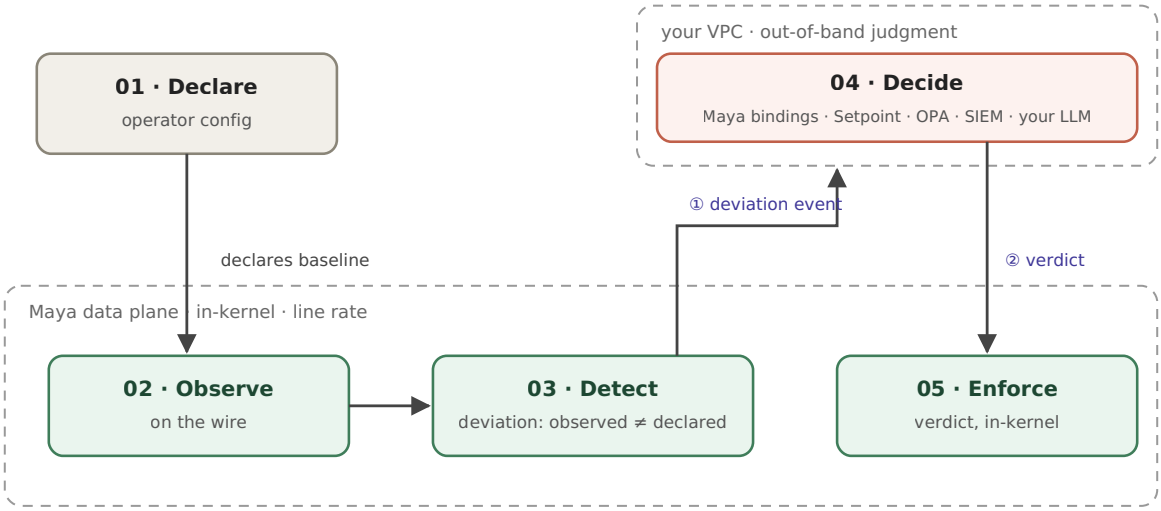
risk. Second, some things a governance program wants (an agent's autonomy mode, the exact model inside a TLS session, the purpose behind a call, the content classification of a flow) are simply **not visible from the envelope**, and Maya names them rather than pretending to check them. The model earns credibility by being exact about its own edges.

5. Bring your own judgment

Here is the line Maya will not cross: **a deviation is not a verdict.**

Maya detects the gap and emits a structured event. What that event *means*, and what to *do* about it, is a decision, and Maya ships with that decision layer in the box: per-type response bindings you set once carry a deviation from observe to alert to contain to drop, and Setpoint, Maya's own monitoring agent (section 8), reasons over drift and recommends action. Plug in nothing, and the loop still runs.

What **bring-your-own-judgment** means is that the verdict seat is open, not empty. If you already have a judgment layer (your OPA policy, your SIEM correlation, an AI agent reasoning inside your own stack), it decides the response and hands it back, and Maya installs it. Your judgment can replace or override Maya's defaults whenever you choose; it is never a prerequisite. The loop:



Decide once per novel deviation → enforce continuously on subsequent traffic

Figure 1. The bring-your-own-judgment loop. Enforcement lives in Maya's data plane, in-kernel, at line rate; evaluation lives out-of-band, in your VPC.

One detail matters more than any other, and we state it plainly because a technical evaluator will ask: **the decision runs out-of-band and is paid once per novel deviation; enforcement thereafter is continuous and in-kernel.** This is a response loop on *subsequent* traffic, not an inline gate on the packet that triggered it. Maya does not stall your data path waiting on a policy engine, and it doesn't pretend to block the one packet that's already gone. It contains what comes next: fast, and forever after, until you say otherwise.

This is why Maya is **the feed first, and the engine only for as long as you want it to be.** It supplies per-agent identity, the live agent-to-agent graph, typed deviation events, and an in-kernel actuator, and its built-in bindings and Setpoint will return the verdicts from day one. The moment your systems want the wheel, they take it. Your judgment, your response logic, your existing stack, your data and your keys: all of that stays yours. Maya complements OPA, your SIEM, and your posture tooling. It is not a replacement for a judgment layer you've already built, and it doesn't try to become a policy language of its own; that would just be a firewall with extra steps.

6. What an operator can declare, and what each buyer's policy wants

Each declaration is one line of deployment configuration, and each maps to a deviation Maya can see from the envelope alone: the destination's network class, the connection direction, the A2A graph, byte volume, timing, the destination's geography. The core set: **Reach** (internal / partner / public), **Peers** (allowed A2A partners), **Providers** (approved model vendors), **Sovereign** (jurisdiction boundary), and **Directionality** (initiates-only vs accepts-inbound). Extended types cover **Environment** (prod / staging / dev), **Dependencies** (expected edges, the one deviation that fires on something that *didn't* happen), **Rate**, **Lifecycle** (max agent lifetime), and **Temporal** (activity windows and change-freezes). A **Budget** type watches egress volume per agent-provider pair.

The point of grouping these under real buyers is that the same envelope-derived model serves very different mandates. Three worked examples.

Financial services: DORA and SOX. The burning issue is proving that your third-party register isn't just a document but an enforced control. Declare **Providers** as exactly the model vendors on your ICT third-party register; the moment an agent's traffic hits an unlisted inference endpoint, Maya emits a *provider-contradiction*: the register, enforced on the wire, a finding an auditor understands instantly. Add **Sovereign** for data-residency, and **Temporal** to make a declared change-freeze window real rather than aspirational. Your GRC engine decides whether that's an alert or a hard stop.

Healthcare: HIPAA. The mandate is minimum-necessary access near protected data. Declare a data-adjacent agent as **Reach: internal-only** and pin its **Providers** and **Peers**. When that agent opens a connection to a public endpoint, Maya emits a *reach-contradiction* before data leaves the boundary, and your engine decides whether to block that agent at its first external packet. This is least-privilege for agents, expressed once and checked continuously.

Internal AI platform: FinOps. Agents call models, and model calls cost money continuously, at machine speed. A misconfigured loop or a prompt-injected agent can run an enormous bill in minutes, and today the first sign is usually the invoice. Declare a per-agent **Budget** and Maya watches measured egress volume against it: WARN, then CRITICAL, as deterministic arithmetic with no model in the loop. In fairness about the mechanism: Maya measures exact byte volume on the wire and applies an operator-supplied bytes-per-token ratio, so the output is a trend-and-ceiling alarm ("*this agent is burning far past its budget*"), not a figure you'd reconcile against a provider invoice. Wire bytes overcount tokens substantially, which is exactly why the operator's empirical ratio beats a fixed constant. Pair it with **Rate** to catch a poll-storm and **Environment** to keep dev traffic out of prod.

Where declarations live: your IAM, not another console. An agent's identity already carries permissions for APIs and data; network rights are the missing column. Maya is extending declarations to attach to the identity systems you already govern agents with: tags on an AWS role, annotations on a Kubernetes ServiceAccount, attributes on a per-agent identity in Entra or GCP. Granted there, an agent's network rights inherit the machinery you already trust: the same commit, the same approval workflow, the same quarterly access review, the same revocation. And where rights haven't been granted yet, Maya can derive a starting profile from the grants your IAM already holds, labeled as derived and promotable to granted with one review. The declaration model doesn't change; what changes is where it's governed. A useful side effect comes free: when the wire shows six distinct agents behaving behind one shared credential, Maya can tell you, and that gap report is a finding your identity program wants regardless.

The regulatory clock is real: the EU AI Act's governance and transparency obligations become enforceable in August 2026, with high-risk record-keeping and monitoring duties following in 2027. The control layer to satisfy them should exist before the deadline, not be assembled at it.

7. The catalog grows with the fabric

Everything above is single-VPC Maya, the standalone product. When Maya goes distributed as the **Global Fabric**, where local enforcement points peer into one governed substrate spanning VPCs, regions, and clouds, the deviation model gains a new class: **global connectivity directives.** You'll declare, and Maya will govern, which agents may connect *across* boundaries, region to region and cloud to cloud. A cross-boundary connection that was never declared becomes a deviation like any other, enforced under one policy authority.

The point for this paper is that deviation is not a single-site trick. It's a model that scales with your topology.

8. Using AI to inspect and govern AI

The declaration catalog is deterministic and envelope-checkable, and deliberately so. But some of the most important deviations aren't a contradiction of a static declaration; they're a departure from an agent's *normal, well-known behavior*, the kind of drift no fixed rule anticipates. That's where you need judgment that reasons, and that's where **Setpoint** comes in.

Setpoint is Maya's monitoring agent, and it works by turning Maya's own architecture on itself: the BYOJ loop, applied recursively. It ingests behavioral telemetry from the mesh over OTLP, and it acts through the northbound API. It is itself a governed agent on the mesh: its own identity, visible in your console, no side door into the data plane, fully out-of-band.

The part that keeps you in control is **BYOLLM: bring your own LLM.** Setpoint doesn't hard-code a model or send your telemetry anywhere. It uses behavioral telemetry to craft dynamic prompts for *your* LLM, running in *your* account, which reasons over deviation-from-normal and instructs Setpoint what to do via the northbound API. Your model, your reasoning, your tenancy. Maya supplies the loop; you supply the mind inside it.

(As the founder, I'll admit I've never loved the name. But the behavior is the point: it watches the setpoint and tells you when an agent drifts off it.)

9. The economics of a hallucination

We know, and we expect, that the LLM will sometimes be wrong. So Setpoint's proposed actions are **human-in-the-loop by default**: it reasons and recommends; a person approves. We are not asking you to hand autonomous blocking to a model that hallucinates.

But look at what we did to the *cost* of that hallucination. In the worst case, a false-positive action is **reversible** and it is **scoped**: Maya's actuator pauses one agent by its identity, surgically, and where even that is too broad it can sever a single conversation or flow and leave the rest of that agent's work running. Not a whole host, not a subnet, not production. The blast radius of being wrong is "one flow cut or one agent paused, undo available, fully audited," not "we took down the service." A mistake that used to be catastrophic is now cheap and recoverable.

And in exchange for accepting that cheap, reversible mistake, look at what the loop does to the clock. The industry benchmark is grim reading: IBM's 2026 Cost of a Data Breach report puts the average time to identify and contain a breach at 247 days, a number that got *worse* this year. With Maya, **time-to-identify is milliseconds**, because a deviation fires on the wire the moment observed contradicts declared, and enforcement, once a verdict lands, is in-kernel and just as fast. What sits between the two is judgment and approval, which is exactly where the time should go. The old loop was a human noticing an alert, opening an investigation, and manually containing a threat, measured in hours if you were lucky. The new loop is telemetry in, reasoning, action out, with a person approving rather than driving: **your MTTC (mean time to contain) drops from hours to seconds.** You've traded a rare, cheap, single-agent false positive for containment fast enough to matter. That is a very good trade, and it's the whole argument for putting AI in the governance loop at all.

10. A Star Trek solution to a Guardians of the Galaxy problem

Maya approaches agentic networking from an AI-native posture from the ground up: identity derived from the wire, deviation as the primitive, bring-your-own-judgment for the verdict, and, where static rules run out, AI reasoning over behavior inside your own tenancy. It can do nearly everything your daddy's firewall can do for east-west and south-north agent traffic. And the things it deliberately can't do, reading payload at the edge and inspecting inbound content, the perimeter still should. Two layers, two jobs, no overlap.

Here's the way I think about it. The perimeter is a Star Trek solution: orderly, deterministic, a clean rulebook for a universe that mostly follows the rules. Agents are a Guardians of the Galaxy problem: a scrappy, autonomous, wildly capable crew that acts on its own and doesn't read your rulebook. You can't apply a Star Trek solution to a Guardians of the Galaxy problem. You need something that sees the crew for who they are, watches what they actually do against what they were declared to be, and hands you the gap as a signal you can govern.

That's Maya.

We're taking on a small number of design partners running agents in their own cloud. If that's you, we'd like to talk.
Mayagentic Inc.