SALT

The Architectural Truth

SALT

The Architectural Truth

Many broad-based security vendors now claim to cover API Security. Unfortunately, simply running traffic through a legacy system (firewall, web apps, etc.), isn't the same as understanding the API Fabric.

Legacy tools were constructed a decade ago. The old language embodied a perimeter-centric world : north-south vs. east-west

Traditional network security was built around a simple mental model:

- **North-south** = traffic crossing the perimeter (user ↔ internet ↔ data center)
- **East-west** = traffic moving laterally inside the environment (service ↔ service)

That model assumed:

- Users sit at the "edge" (Users are outside, assets are inside)
- Apps live inside a fixed perimeter, a defensible wall
- Traffic patterns are mostly predictable and linear

It worked well enough when most flows were web browsers talking to web apps over a few well-known paths.

APIs (and agents) shattered that model

APIs turned every service into both a client and a server. Agentic AI turned every **LLM, MCP server, tool, and SaaS app** into an active participant in your environment.

In this world: a single user request triggers a complex, multi-directional chain.

- A **user** calls an **AI assistant**
- That assistant calls an **MCP server**
- The MCP server fans out to:
  - Internal APIs and microservices
  - Databases and data lakes
  - SaaS tools (CRM, ticketing, storage, payments)
  - Other agents and automation platforms

Then:

- Those services call back into **webhooks**,
- Trigger **workflows in third-party SaaS**,
- Fire events into **queues and event buses**,
- And sometimes call back into the **original agent** or **another agent**.

Which direction is that?
It's not "north-south" or "east-west" – it's **every direction at once**.

The result is a **dynamic multi-directional API fabric**:
- **Multi-hop:** One request becomes dozens of downstream calls
- **Multi-cloud / multi-SaaS:** Boundaries between "inside" and "outside" are blurry
- **Multi-actor:** Humans, services, agents, schedulers, and third-party systems all drive traffic
- **Dynamic:** New APIs, tools, and agents appear and change behavior continuously

Trying to reason about this with static compass points is like trying to describe a spider web as "just a horizontal line."

The API fabric is multi-directional by design

In the API fabric, *every node is both client and server*:

- An **AI agent** is a client of the MCP server, and a server for chat/webhook APIs.
- A **microservice** is a server for other services, and a client of databases, queues, and SaaS APIs.
- A **SaaS platform** is a server for your webhooks, and a client of your internal APIs.

Security-critical flows now look like:

- An external prompt → LLM → MCP server → sensitive internal API
- A compromised SaaS token → Third-party app → your webhook → internal automation APIs
- An internal agent → external tool API → data exfil path you never modeled as "north" or "south"

Attackers don't care about your compass. They care about paths: the fastest chain of APIs that leads to sensitive data or dangerous actions.

Why visibility must be multi-directional

Because the fabric is multi-directional, **point solutions that only see traffic at the edge or at a single choke point are blind to most of the story:**

What Legacy Tools See (The Old Model)	What They Are Blind To (The API Fabric)
Endpoint (North-South):- Sees initial user request and final response	**The Critical Middle:** Internal API calls, agent-to-agent hops, SaaS-to-webhook flows, cross Saas and Multi-cloud paths
The Internal Network (East-West): Sees raw traffic between nodes	**The Context:** API semantics, payloads, AI/Agent interactions.
Snapshot: A single point in time view of the network	**Dynamic Behavior:** How the fabric evolves as new API's tools, and agents come online.
	Identity & Authorization: Caller type (user/service/agent/SaaS), auth method (tokens, scopes), least-privilege policy decisions
	Data Sensitivity & Governance: PII/PHI/secrets classification, redaction, residency/sovereignty tags
	Inventory & Change: Continuous/API discovery, versioning, newly exposed endpoints, deprecations

- Edge and WAAP controls see **some ingress/egress** but miss:
    - Internal tool calls
    - Agent-to-agent hops
    - SaaS-to-webhook flows
    - Background system-initiated traffic
- Internal network tools see **some service-to-service** calls but:
    - Don't understand API semantics, payloads, or AI/agent context
    - Don't see how external SaaS and agents chain into those APIs

To actually secure APIs and AI, you need to see:

A. **Every caller** (user, service, agent, SaaS, scheduler)
B. **Every call** (internal, external, cross-cloud, cross-SaaS)
C. **Every direction** (ingress, egress, lateral, and the weird "off-axis" paths agents create)
D. **Every behavior pattern** over time (what's normal vs. abuse, data exfil, or agent misuse)

That's what we mean by **multi-directional visibility across the API fabric**.

Why Salt is built for the API fabric (and for agents)

Salt is designed around the fabric, not the perimeter:

- We **observe real API traffic** everywhere it runs – across clouds, gateways, service meshes, and SaaS-integrated flows.
- We rebuild the **true API fabric** from that traffic:
    - Every endpoint
    - Every consumer (including agents and tools)
    - Every data path and behavior pattern over time
- We detect and stop:
    - Abuse and data exfil that happens inside the fabric
    - Malicious or misconfigured agents chaining APIs in unsafe ways
    - Risky API exposure created by MCP servers and new tools
    - Attacks that never look like “classic” edge or lateral movement, but do look like abnormal API behavior.

In other words:
Only technologies that see the fabric as multi-directional – like Salt – can actually solve the API and AI security problem.

If you can’t see the full mesh of calls agents are making, you’re not securing agentic AI. You’re securing a 10-year-old perimeter model that no longer exists.