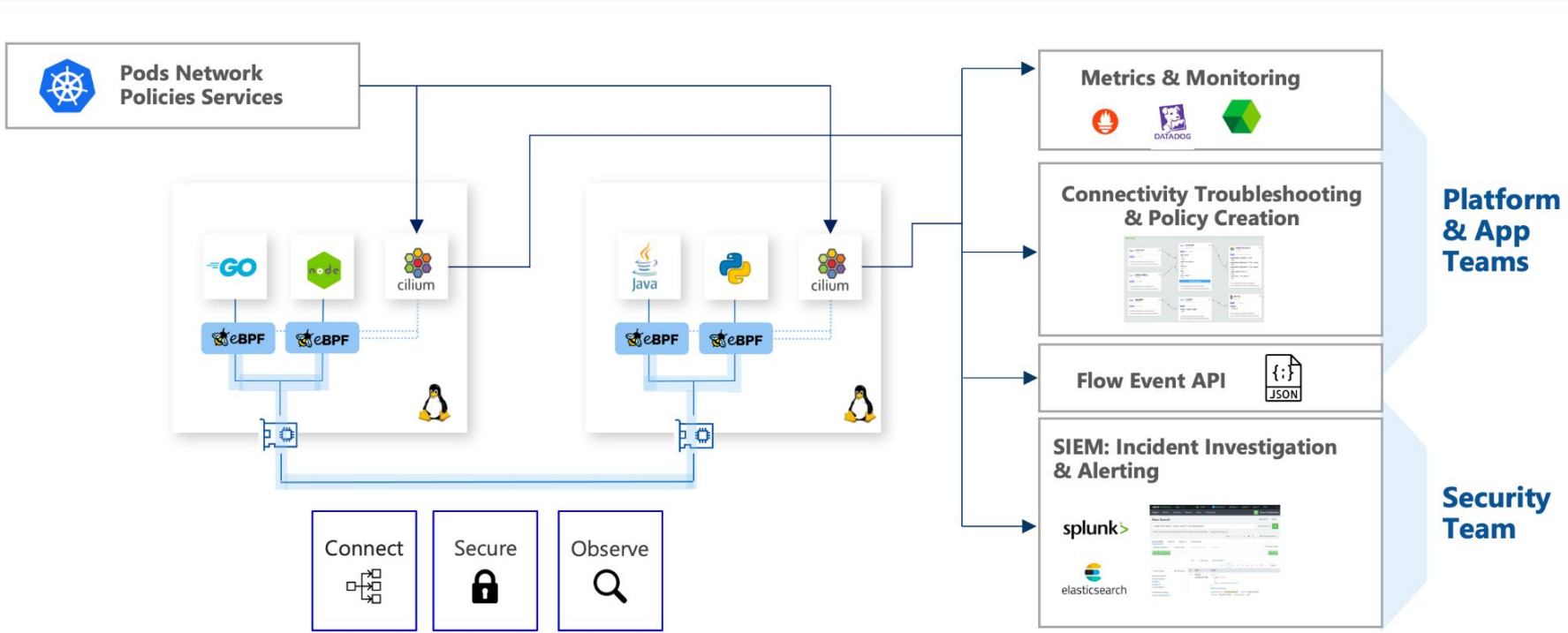
Enforcing MCP Tool Call Safety Outside the Model with Cilium & Tetragon

Defense-in-Depth for AI Agents on Kubernetes

MCP makes it easy to give LLMs hands and feet (tools/call). When an agent is tricked into calling a valid tool with malicious intent, deterministic enforcement must stop it across the protocol, network, and Linux kernel layers.

tetragon.io

tetragon



cilium.io

tetragon


tetragon
Metrics
Events
SIEM
fluentd
Logs
Traces
JSON
Tetragon Agent
Pod
app.py
Func Calls
Pod
app.go
Code Exec
Linux Kernel
Kernel Runtime
tetragon
Smart Collector
Stack Traces
Ring Buffer
Metrics
Hash Maps
Process Execution
Syscall Activity
System Calls
File Access
VFS
Seq Attack
TCP/IP
NS Escapes
Priv Escalations
Namespaces
Data Access
Storage
HTTP, DNS, TLS
eth0
Network

tetragon.io

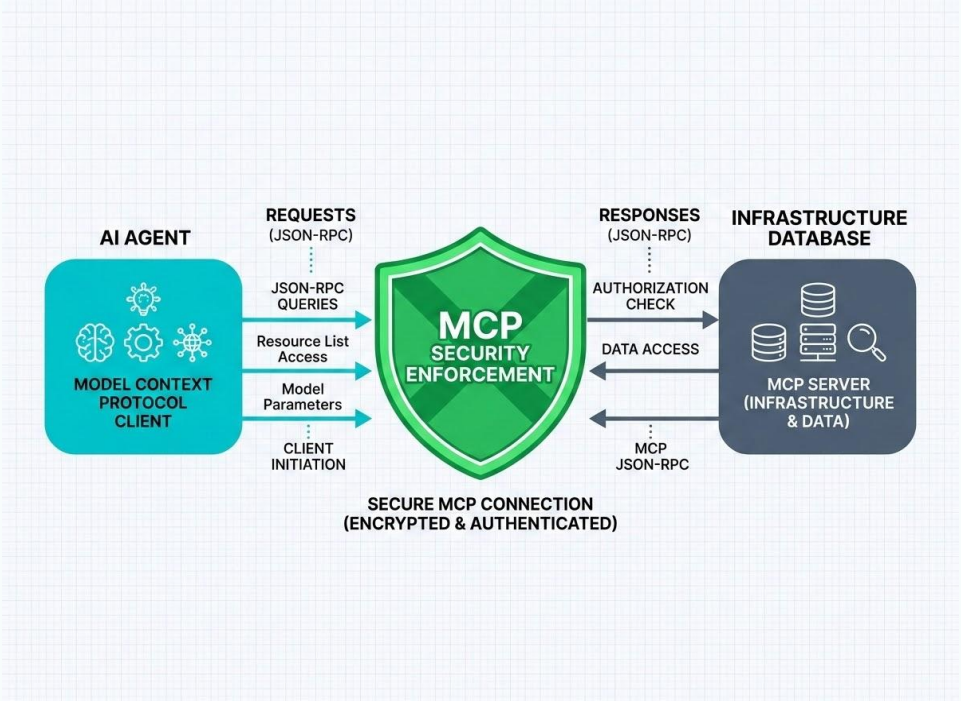
The Problem: Prompt Guardrails Inspect Text, Not Execution

Poisoned context (e.g. hidden prompt injection in tickets, webpages, or issues) leads the LLM to emit a syntactically valid **JSON-RPC tools/call** payload.

To the MCP server, `{"method": "tools/call", "params": {"name": "drop_database"}}` looks completely legitimate and authentic.

Text-based guardrails inside the model loop are probabilistic.

Effective enforcement need to be **deterministic and outside the model**.



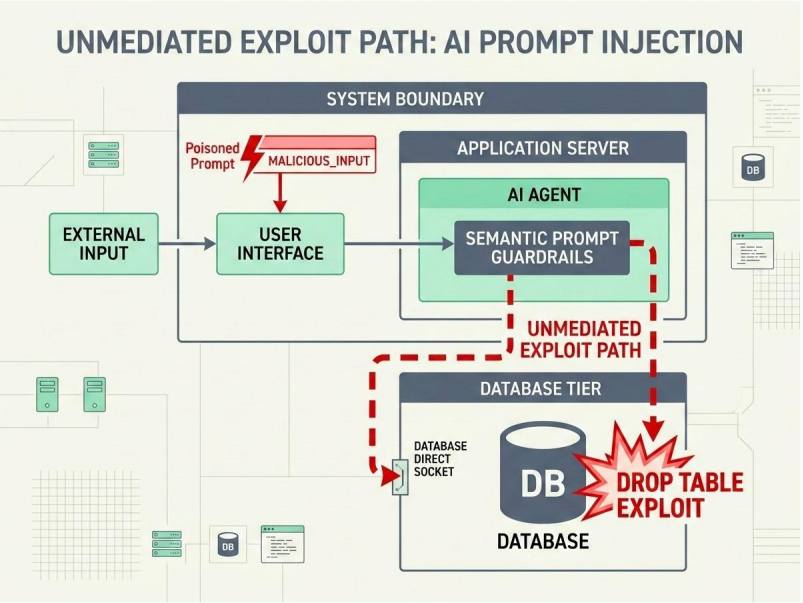
tetragon.io

tetragon

What Can Go Wrong Inside an MCP Tool Runner?

Running agent tools introduces three direct risks that prompt filters cannot stop:

- **1. Dangerous Actions:** The model can call destructive functions like drop_database without human approval.
- **2. Data Leaks:** A compromised runner can send internal tokens, database rows, or environment variables to external servers.
- **3. Shell Escapes:** Unsanitized tool inputs can run host commands, launching /bin/sh or opening a reverse shell.

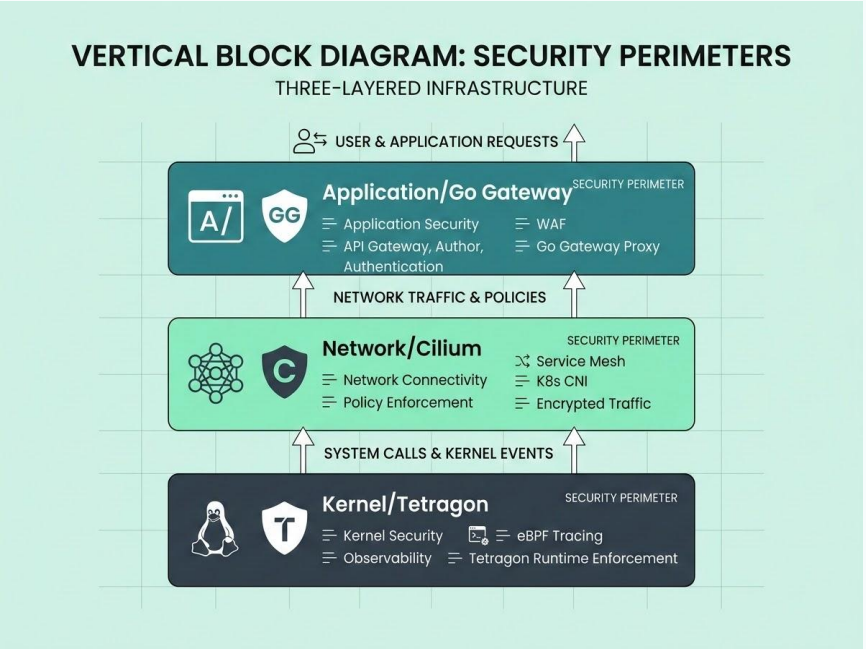


Architecture: 3-Layer Defense-in-Depth

🛡️ **Layer 1 (Application / Intent):** Go proxy intercepts JSON-RPC, inspects params.name, and places high-risk/unknown tools into PENDING_APPROVAL state.

🌐 **Layer 2 (Network Boundary):** Cilium enforces identity-aware L3/L4 egress rules, locking traffic strictly to kube-dns and approved-backend:8080.

🔒 **Layer 3 (Kernel Runtime):** Tetragon hooks sys_execve via eBPF tracepoints in the kernel, sending synchronous SIGKILL before shell binaries can initialize.

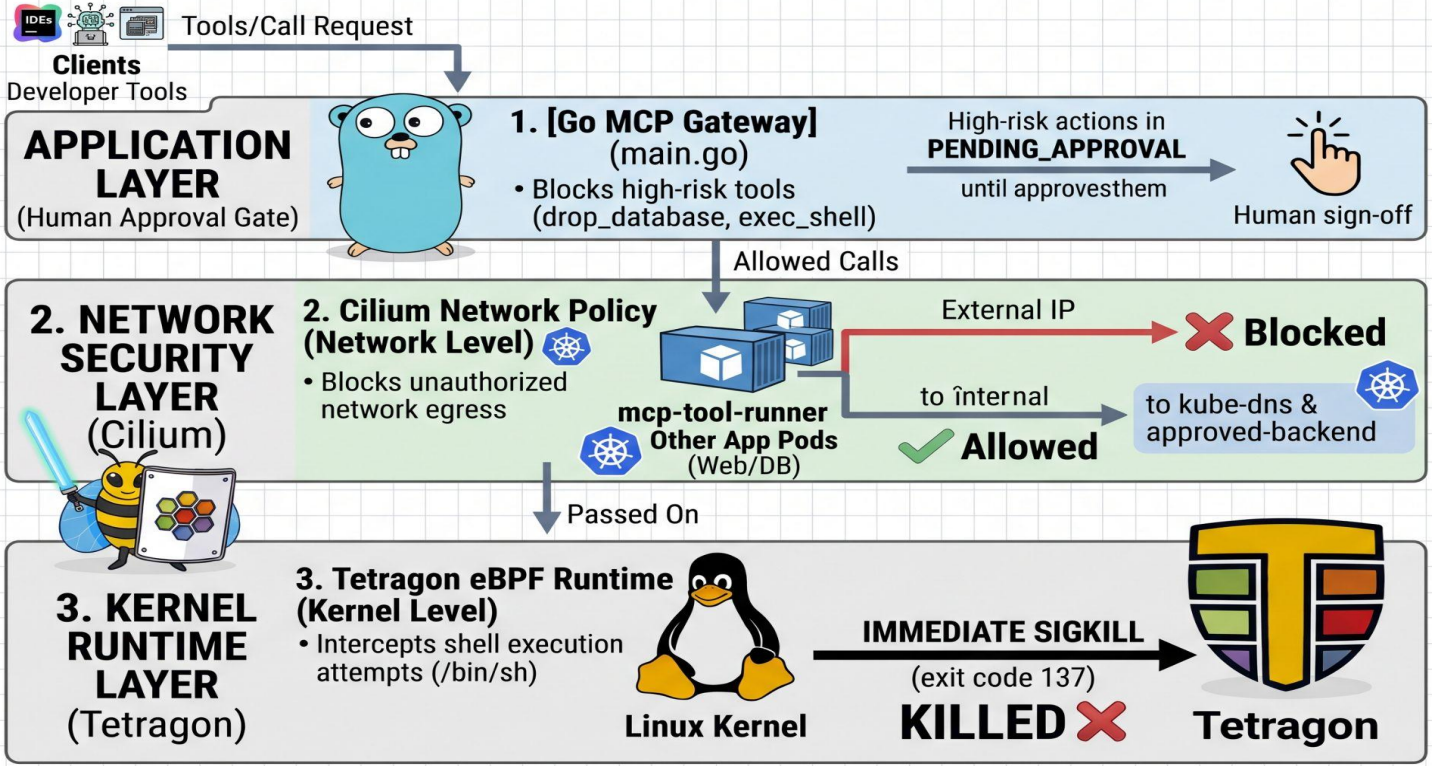


tetragon.io

tetragon

MCP TOOL CALL ENFORCEMENT WITH TETRAGON & CILIUM

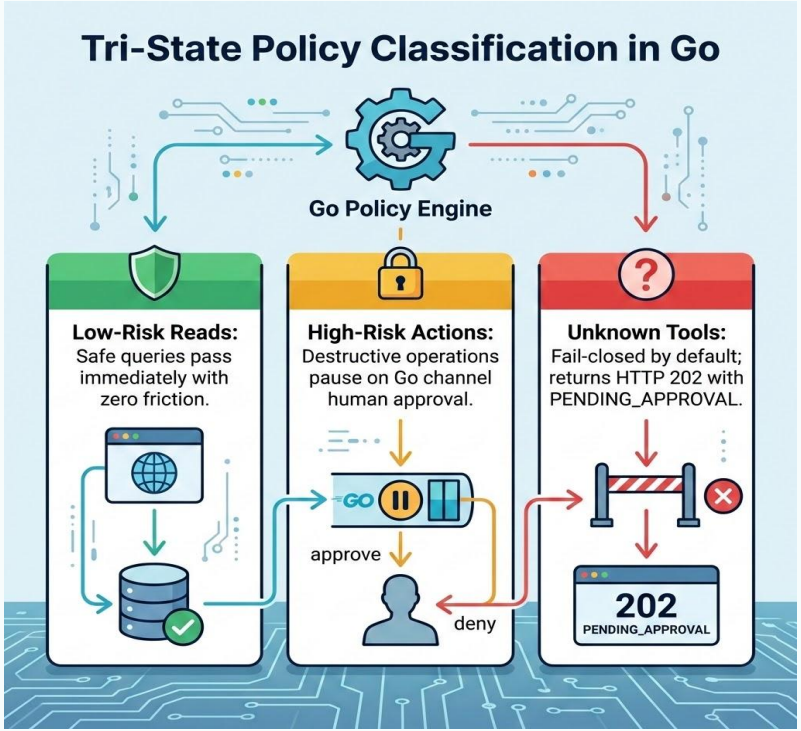
Defense-in-depth enforcement for Model Context Protocol (MCP) tool calls on Kubernetes



Deep Dive: Layer 1 (Go Approval Gateway)

Tri-State Policy Classification in Go:

- **Low-Risk Reads:** Safe queries pass immediately with no friction.
- **High-Risk Actions:** Destructive operations pause on Go channel awaiting human approval.
- **Unknown Tools:** Fail-closed by default; returns HTTP 202 with PENDING_APPROVAL.



Deep Dive: Layers 2 & 3 (Cilium + Tetragon eBPF)

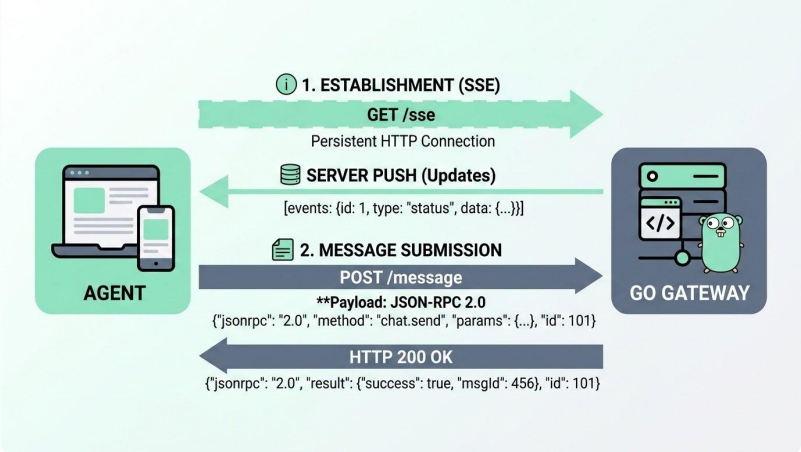
eBPF Datapath Enforcement in the Linux Kernel:

🌐 Layer 2 — Cilium Network Policy (cnp-mcp-egress.yaml):

Enforces strict L3/L4 egress; locks outbound traffic to approved-backend:8080 and kube-dns, preventing data exfiltration.

🔒 Layer 3 — Tetragon TracingPolicy (tp-block-shell.yaml):

Hooks kernel sys_execve via eBPF; intercepts [/bin/sh, /bin/bash] and delivers synchronous Sigkill before shell init.



Live Demo: Breaking & Hardening mcp-demo

A **Pod 1**

mcp-gateway

Go approval proxy receiving SSE connections and HTTP messages. Inspects tools/call payloads and holds sensitive calls in escrow.

B **Pod 2**

mcp-tool-runner

The execution sandbox running agent tool logic. Targeted by Cilium network policies and Tetragon kernel tracing to prevent exfiltration & shell escape.

C **Pod 3**

approved-backend

The protected internal service on port 8080. Validated requests flow through; direct unmediated access from unauthorized pods is rejected at L4/L7.

Productionizing: From Lab to Real Clusters

1. Start in Audit Mode: Log normal tool activity with Tetragon first before turning on automatic blocking.

2. Lock Down Containers: Use read-only filesystems and minimal container images without unnecessary binaries.

3. Dynamic Approvals: Route high-risk requests to team webhooks or policy engines like OPA.

Production Operational Principle

Enforcement outside the model provides predictable guarantees: network policies and kernel probes operate independently of model hallucination or prompt injection.

Try It Yourself: Interactive Lab & GitHub Repo

💻 Killercoda Interactive Lab:



🛠️ GitHub Repo:



🛠️ Instruqt Isovalent labs:



tetragon.io

tetragon