hoop.dev

TECHNICAL GUIDE

AI Agent Infrastructure Security

Claude Code Gateway

Control what AI sees and does in production infrastructure.

- Mask PII, secrets, and credentials before they reach the model
- Authenticate through SSO. Database credentials never touch Claude Code.
- Log every query at command level with full context
- Block or gate dangerous actions with inline guardrails

150+	<50ms	30+	0
PII Types Detected	Masking Latency	Protocols Supported	Stored Credentials

CONTENTS

01

The Problem

The threat model shifted

Give a coding assistant API access and it is no longer making suggestions. It reads production data. It ingests environment variables. It operates in the same execution path as your engineers, except it never forgets what it read.

Most teams respond by sandboxing the model: strip the production context, lock down access, accept that it will be less useful. That tradeoff feels necessary. It is not.

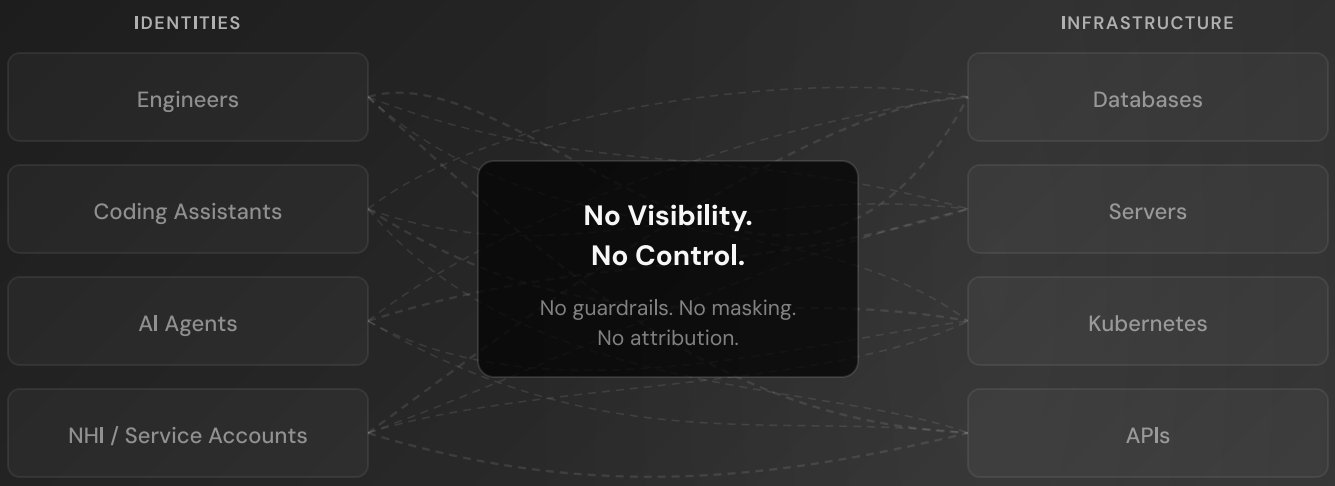
Session-level controls break

When a human encounters a secret in a terminal, they move on. The secret lives briefly in short-term memory and then it is gone. It does not get parsed, cached, or sent across an API boundary.

Claude Code does not move on. It reads source code, config files, stack traces, terminal output, and API responses. It pulls all of it into its context window. Every secret a human would have ignored becomes structured, machine-readable input.

> **The boundary must move**
>
> Stop thinking about **who can connect**. Start thinking about **what the model can see, run, and send**. Identity alone is not the right boundary anymore.

Without a proxy sitting between identities and infrastructure, every connection is a direct line. No visibility, no guardrails, no masking, no reliable attribution.



02

Three Converging Risks

1. Prompt Injection

A malicious string hiding in a database field, a log entry, or an API response can quietly redirect the model. One bad input changes what Claude Code does next.

2. Sensitive Data Exposure

API keys, database passwords, customer PII, tokens buried in environment variables: all of it gets pulled into the model's working memory the moment it is readable.

3. External Exfiltration

Once a secret enters the context window, it can travel. It shows up in API calls, code suggestions, outbound requests. By the time you notice, the data has already left.

Protocol-level mitigation

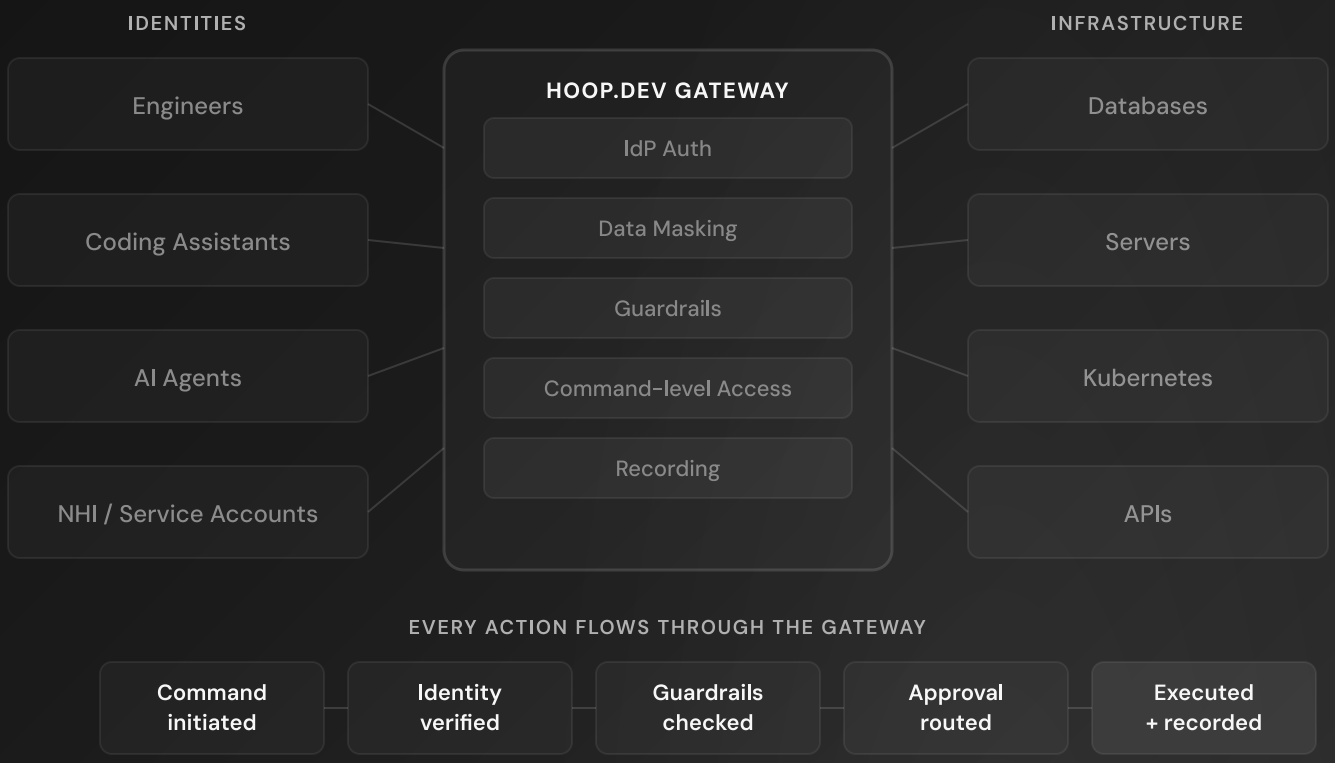
hoop.dev addresses all three at the network boundary, before any data reaches the model.

RISK	HOOP.DEV CONTROL	OUTCOME
Prompt Injection	Request and response inspection, policy guardrails, and approval gates at the HTTP layer	Bounded execution with reviewable decision points
Data Exposure	Real-time masking and response filtering before data reaches Claude Code	Secrets never enter the model context
Exfiltration	Outbound allowlisting, approval gates, and full audit logging	Controlled egress with provable containment

03

Architecture

hoop.dev sits between Claude Code and your infrastructure as a protocol-level proxy. It is stateless by design. No credentials are stored in memory, on disk, or anywhere in between.



Data flow

1. **Query initiation**
   Command sent through the hoop.dev proxy
2. **SSO authentication**
   OpenID Connect confirms identity
3. **Protocol interception**
   Gateway parses at wire protocol level
4. **Policy evaluation**
   Guardrails check rules. Block or gate.
5. **JIT credential retrieval**
   Agent pulls creds from secrets manager
6. **Execution**
   Query runs against the target system
7. **Response masking**
   ML masks 150+ PII types before delivery
8. **Audit capture**
   Command, identity, and redaction logged

Security Controls

AI data masking

Sensitive data gets caught in transit, not after the fact. ML-powered detection masks PII at the protocol layer before the response reaches Claude Code. There are no post-download scans and no manual rules to configure.

Coverage spans 150+ PII types across 40+ countries: financial identifiers, government IDs, health records, auth tokens, network metadata, and biometric markers.

Masking strategies

STRATEGY	EXAMPLE	USE CASE
Full redaction	[REDACTED]	Passwords, auth tokens, encryption keys
Partial masking	****_****_****-1234	Credit cards, SSNs, phone numbers
Hashing	SHA256(value)	User IDs, session identifiers
Format preserving	Structure stays, values change	Dates, numeric IDs for testing

Credential offload

Claude Code authenticates through SSO. It never sees a database password. The agent pulls credentials just-in-time from your secrets manager, uses them for the session, and discards them when the session ends. Nothing is stored in the gateway. Not temporarily, not at all.

Guardrails

Every command gets checked before it runs. If it breaks a rule, it is blocked. If it needs a second pair of eyes, it enters an approval workflow. The model does not decide. Your policies do.

Runbook validation

Pre-built query templates where every parameter is checked against a pattern. Commands stay inside approved shapes. Injection does not get through.

Approval workflows

Risky commands trigger a Slack notification to the right approvers. They see the exact query, not a vague access request. Multi-stage approval for high-risk actions.

Connection segmentation

One connection per risk level: prod-db-readonly, prod-db-write, prod-db-admin. Each with its own rules, its own approvers, its own guardrails.

Outbound allowlisting

Only approved destinations receive outbound traffic. Every request is logged. Nothing leaves your environment without a record of where it went and why.

Audit trail

Every query is logged individually. Not the session, the query. Each entry carries 15+ fields: event ID, timestamp, user identity, source IP, target resource, command type, full payload, approval chain, and what was redacted.

Logs are append-only and signed with SHA-256, so they cannot be altered after the fact. Export to Splunk, Datadog, or Elastic via webhooks.

> **Actions, not sessions**
>
> Traditional PAM tools record sessions. hoop.dev records actions. The difference matters: you can search for "every DELETE on production this month" instead of scrubbing through hours of screen recordings looking for something you might not find.

05

Model Security vs Execution Security

Anthropic builds security controls into Claude Code. Those controls shape how the model behaves after it receives your data. They are real, and they matter.

But consider the sequence: you send sensitive data to an external provider so the model can evaluate whether that data is sensitive. By the time it reaches that conclusion, the data has already crossed the trust boundary.

Claude Code security tells you what happened. hoop.dev stops it from happening.

RESPONSIBILITY	CLAUDE CODE (MODEL)	HOOP.DEV (PROTOCOL)
Model reasoning integrity	Built-in controls that shape behavior after context is received	—
Prompt injection controls	Model-level detection and safeguards	Request inspection, guardrails, and approval gates before execution
Secret masking before ingestion	—	Real-time filtering at the HTTP layer. Secrets never reach the model.
Outbound egress enforcement	—	Allowlisting, approval gates, and logging of all outbound traffic
Deterministic audit trail	Covers model-side events only	Full request and response traceability at the network layer

> **Complementary, not competing**
>
> The model governs its own reasoning. The proxy governs what gets in and what gets out. You need both layers working together. Neither one is the whole story.

The protocol layer closes the gap

Mask secrets before the model sees them. Enforce policy on every command. Route outbound traffic through a proxy that logs everything. What you get is a Claude Code that operates with real context, inside a boundary you can measure and audit.

06

Compliance Evidence

hoop.dev holds SOC 2 Type II certification and is GDPR compliant. It is MIT licensed and a CNCF member. For additional frameworks, the platform generates structured evidence mapped to specific controls, automatically, as part of normal operation.

SOC 2	HIPAA	PCI-DSS	SOX	NIST	GDPR
18	12	5	8	38	6
controls addressed	controls addressed	controls addressed	controls addressed	controls addressed	controls addressed

Session recordings

Every session captured with identity, timestamps, commands, and responses. Stored immutably. Retention from 90 days to 7 years.

Masking logs

Every PII field detected and redacted, tracked by count, type, and source. Ready for audit review without additional preparation.

Approval chains

Every request, every reviewer, every decision, every timestamp. Full chain of custody from the moment access is requested to the moment it completes.

Guardrail events

Every blocked command with context: who ran it, what it was, when it happened, and which policy caught it.

> **Evidence, not certification**
>
> HIPAA, PCI DSS, NIST, and SOX numbers above reflect controls that hoop.dev generates evidence for. They are not certifications hoop.dev holds. SOC 2 Type II and GDPR compliance are confirmed.

07

Threat Model

These are the attack vectors that show up when AI coding agents have production access. Each one maps to a specific control in the gateway.

ATTACK VECTOR	HOW IT HAPPENS	HOOP.DEV CONTROL
Credential theft	The model reads .env files, API keys, or database passwords from terminal output	Zero stored credentials. JIT retrieval from your secrets manager. Keys never enter the context window.
PII exfiltration	The model pulls customer data from query results and sends it in subsequent API calls	AI data masking redacts 150+ PII types before the response reaches the model.
Prompt injection via data	A malicious string in a database field or log entry changes what the model does next	Request and response inspection plus policy guardrails filter payloads before they reach the model.
Privilege escalation	The model runs DROP TABLE, modifies user privileges, or executes destructive writes	Guardrails block or gate those commands. Approval workflows put a human in the loop.
Lateral movement	The model uses credentials from one system to access another	Each connection has its own policies. No shared credentials. No network-layer pivot.
Insider threat	An authorized user leverages Claude Code to pull data at scale	PII is masked even for authorized users. Every action is logged with full context and attribution.

> **Defense in depth**
>
> No single control covers all of these. hoop.dev layers six: SSO authentication, role-based authorization, protocol interception, real-time data masking, immutable audit logging, and JIT credential management. They work because they overlap.

If Claude Code can call your APIs, enforcement has to live inline.

hoop.dev routes Claude Code's API traffic through a protocol-aware proxy. It masks data, isolates credentials, enforces guardrails, and logs every action. The result is production access with boundaries you can see, measure, and prove.

Documentation: hoop.dev/docs

GitHub: github.com/hoophq/hoop

License: MIT

Contact: sales@hoop.dev