ANTIHERO
Technical Whitepaper

Distributed Alignment Architecture for Agentic AI
Government-Ready Security, Insurance, and Compliance

VERSION 3.0 | FEBRUARY 2026
antihero.dev

This document describes the architecture, capabilities, and formal properties of Antihero—a distributed alignment layer that enforces policy at the action boundary of AI agent systems. Version 3.0 introduces 17 new capabilities spanning advanced intelligence, government-grade security, and enterprise-ready operations. 787 automated tests. Zero regressions.

What's new in v3.0: Trajectory analysis, canary tokens, semantic DLP, explainable denials, formal verification, policy knowledge graphs, incident response automation, policy version control, compliance mapping (7 frameworks), multi-tenant hierarchy, agent observability with drift detection, FIPS cryptography & air-gap deployment, federated policy sync, policy marketplace, and digital twin simulation.

Table of Contents

Part V — Formal Foundations

Appendices

Part I — Foundation

1. The Structural Shift: From Language Safety to Action Safety

The dominant approach to AI safety treats the problem as a language problem: align what the model *says*. RLHF, constitutional AI, red-teaming, output filtering—these techniques target the generation boundary. They ask: *did the model produce harmful text?*

This framing was adequate when AI systems were conversational. It is inadequate when AI systems *act*. An AI agent that books flights, writes code, queries databases, sends emails, and manages infrastructure does not merely generate tokens. It executes side effects in the real world. The risk surface has moved from the output boundary to the **action boundary**.

Antihero is built on a simple thesis: **the correct place to enforce alignment is where actions happen, not where words are generated.** Every tool call, every file write, every API request, every database query—each is a concrete action that policy can intercept, evaluate, and either permit or deny. This is the action boundary.

> *"What does it mean for a machine to want something? Not what it says it wants—what it does." — Brian Christian, The Alignment Problem*

Antihero is not a model. It is not a fine-tune. It is not a prompt injection defense. It is a **security and insurance layer** that wraps around any AI agent or chat interface, enforcing policy at the action boundary with cryptographic evidence and optional liability coverage. Think Norton Antivirus for AI: a protection layer that sits between the agent and the world.

2. Threat Model: Agentic Autonomy as Attack Surface

We model six adversary classes against AI agent deployments:

A1: Prompt Injection (Direct & Indirect). Adversarial instructions embedded in user input or fetched content that hijack agent behavior. Antihero mitigates this by evaluating *actions*, not

prompts—even a compromised model must pass policy evaluation before executing any side effect.

A2: Tool Misuse. Agents calling tools in unintended ways: writing to production databases, exfiltrating data via API calls, executing shell commands. Policy rules match on action + resource + context to block unauthorized tool usage.

A3: Privilege Escalation. Agents acquiring capabilities beyond their designated scope through delegation chains, role confusion, or permission drift. Delegation chains enforce monotonic capability attenuation.

A4: Data Exfiltration. Agents leaking PII, secrets, or sensitive data through tool calls, API responses, or file operations. Content inspection scans every field for PII patterns (SSN, credit card, API keys) pre- and post-execution.

A5: Supply Chain Compromise. Malicious or backdoored tools, plugins, or MCP servers introduced into the agent's tool set. Policy evaluation applies regardless of tool origin; canary tokens detect unauthorized data access.

A6: Behavioral Drift. Agents gradually shifting behavior over time as context windows, fine-tuning, or environmental changes alter their action patterns. Per-agent behavioral baselines detect frequency, temporal, and action-type anomalies.

The assets at risk include: user data, system configuration, financial resources, infrastructure access, reputation, and regulatory compliance. Antihero's threat model assumes that **any** component in the agent stack may be compromised—including the model itself—and enforces policy at the last trusted boundary before execution.

3. Architecture: The Three-Layer Stack

Every AI action in Antihero passes through three layers: Policy, Enforcement, and Evidence.

Layer 1: Policy (Antihero Policy Specification)

Policies are declarative YAML rules that match on **subject** (who), **action** (what), **resource** (where), and **conditions** (when/how). Three effects are supported: deny, allow, and allow_with_requirements. Requirements include human confirmation, MFA, content redaction, sandbox isolation, and rate limiting.

Policies compose across four tiers: **baseline** (Antihero defaults), **org** (organization-level), **app** (application-level), and **user** (per-user overrides). Composition follows a strict monotonic safety rule: **deny always dominates**. A deny at any tier cannot be overridden by an allow at a higher tier.

Requirements merge via union.

Layer 2: Enforcement

Tool adapters emit a standardized **Tool Call Envelope (TCE)** containing the action, resource, subject identity, and contextual metadata. The policy engine evaluates the TCE against all applicable rules and returns a **Policy Decision Envelope (PDE)** with the effect, matched rule, risk score, and any requirements.

Enforcement is **fail-closed**: if the policy engine is unreachable, crashes, or times out, the default effect is deny. No side effect executes without explicit policy evaluation. The enforcement pipeline includes 16 discrete steps: rate limiting, content inspection, threat scanning, policy evaluation, requirement handling, execution gating, chain append, latency recording, observability, and incident management.

Layer 3: Evidence

Every evaluation produces an **Audit Event Envelope (AEE)**—a hash-chained, append-only record containing the TCE, PDE, execution outcome, timestamps, and cryptographic chain link. AEEs use RFC 8785 JCS canonicalization for deterministic serialization and support Ed25519 signatures for enterprise deployments.

The evidence chain is tamper-evident: each AEE’s hash includes the previous AEE’s hash, forming an append-only Merkle-like structure. Any tampering with historical records breaks the chain and is immediately detectable. This makes the audit trail insurance-grade—suitable for regulatory audits, compliance certificates, and claims adjudication.

Part II — Core Capabilities

4. Content Inspection and Data Loss Prevention

Every field in every TCE is scanned for sensitive content before and after execution. The content inspector detects:

- Social Security Numbers (SSN) — XXX-XX-XXXX patterns with Luhn-like validation
- Credit card numbers — major network prefixes with Luhn check
- API keys and tokens — AWS, GitHub, Stripe, and generic patterns
- JWTs and bearer tokens — header.payload.signature structure
- Private keys — PEM/PKCS8 headers
- Email addresses, phone numbers, IP addresses
- Custom patterns — user-defined regex rules

Findings are classified by severity (critical, high, medium, low) and type. When a policy requires redaction, the content inspector replaces sensitive values with masked tokens ([REDACTED:SSN]) before the action executes. Post-execution inspection catches data returned by tools.

5. Behavioral Analytics and Anomaly Detection

Antihero maintains per-agent behavioral baselines that track action frequency, resource access patterns, temporal distribution, and action-type mix. Four anomaly types are detected:

- **Frequency anomaly** — Action rate exceeds 3× the rolling average
- **Unseen action** — Agent performs an action type never seen in its baseline
- **Temporal anomaly** — Agent active outside its normal operating hours
- **Resource deviation** — Agent accesses a resource not in its historical profile

Anomaly flags are attached to TCEs and feed into policy evaluation. Policies can match on anomaly type and severity, enabling rules like “deny all unseen actions from production agents” or “require MFA for temporal anomalies.”

6. Agent Delegation and Identity

Agents can delegate work to sub-agents via delegation chains. Each delegation hop attenuates capabilities: the child agent can never have more permissions than its parent. Delegation chains are signed with Ed25519 keys and carry full lineage metadata.

Configurable depth limits (default: 5) prevent unbounded delegation trees. Cross-organization identity federation uses a trust registry where organizations register public keys. Agent identities are verified cryptographically at trust boundaries.

7. AI Liability Insurance

Antihero is not only a policy engine—it is **underwriting infrastructure**. The same cryptographic receipts that enforce policy also back insurance claims. This creates a flywheel: better security posture leads to lower premiums, which incentivizes stronger enforcement, which reduces risk.

Risk scoring uses six dimensions: Capability Surface (CS), Autonomy Depth (AD), Network Openness (NO), Memory Persistence (MP), Human Gating Strength (HG), and Audit Integrity (AI). Each dimension produces a 0–1.0 score; the composite risk score drives premium calculation through transparent multipliers.

Claims are adjudicated against the cryptographic evidence chain. When an insured agent causes damage, the AEE chain provides deterministic proof of what happened, when, and which policies were in effect. Reinsurance treaty modeling supports Quota Share, Excess of Loss, and Hybrid structures.

8. Compliance and Regulatory Alignment

Antihero generates compliance certificates with SHA-256 integrity hashes and scored findings. Built-in support for:

- EU AI Act — High-risk system requirements, human oversight, transparency
- NIST AI RMF — Govern, Map, Measure, Manage framework alignment
- SOC 2 Type II — Security, availability, and confidentiality controls
- HIPAA — PHI safeguards and audit requirements

See Section 20 for the extended Compliance Mapping Engine with 7 framework templates and automated gap analysis.

9. Agent Plan Graphs: Trajectory-Level Alignment

Individual action evaluation is necessary but not sufficient. An agent can execute a sequence of individually-allowed actions that, taken together, constitute an attack (e.g., read credentials → list databases → exfiltrate data). Agent Plan Graphs (APGs) model trajectories as directed acyclic graphs of goals, subgoals, tool calls, and memory writes.

APG nodes carry risk annotations. Policy evaluation can match on trajectory patterns as well as individual actions. Risk budgets constrain the total risk an agent can accumulate before requiring human intervention. See Section 10 for the Trajectory Analysis engine that implements APG-level threat detection.

Part III — Advanced Intelligence

Part III describes seven capabilities that were added in response to competitive analysis and security researcher feedback. These features represent Antihero's transition from a policy enforcement engine to a comprehensive AI security intelligence platform.

10. Trajectory Analysis: APG-Level Threat Detection

The Trajectory Analyzer detects multi-step attack patterns by maintaining per-agent action windows and matching them against known threat signatures. Unlike individual action evaluation, trajectory analysis considers the *sequence* of actions.

Five built-in patterns are included: **credential harvesting** (read secrets → network access), **data exfiltration** (list → read → send), **privilege escalation** (read config → write config → execute), **reconnaissance** (systematic listing of resources), and **persistence** (schedule → write → create user). Custom patterns can be defined with action sequences, confidence thresholds, and time windows.

When a trajectory match is detected, the Guard pipeline automatically creates an incident (see Section 18) and can quarantine the offending agent. Trajectory analysis integrates with the policy engine: trajectory risk scores feed into policy conditions, enabling rules like "deny all actions from agents with active trajectory matches."

11. Canary Tokens: Active Defense at the Data Layer

Canary tokens are decoy data elements planted in resources that agents can access. When an agent reads or transmits a canary token, it triggers an alert—proving that the agent accessed data it should not have, or that data is being exfiltrated.

Antihero supports seven token types: **API keys**, **database credentials**, **PII records** (fake SSNs, emails), **file paths**, **URLs**, **DNS names**, and **custom patterns**. Tokens are generated with cryptographic uniqueness and include metadata (deployment location, creation time, expected access pattern).

The CanaryRegistry scans TCE fields (action arguments, resource content) for deployed tokens. Detections produce CanaryAlert objects with severity, agent identity, and access context. Alerts integrate with the incident response system and notification channels. Token rotation and lifecycle

management are supported.

12. Latency Management and Decision Caching

Policy evaluation must be fast enough that it does not meaningfully slow down agent execution. Antihero's latency management system provides:

- **LatencyTracker** — Rolling window (default 1000) of per-action latency measurements with percentile reporting (p50, p95, p99)
- **DecisionCache** — LRU cache with configurable TTL (default 60s) and max size (default 1000). Cache keys combine action + resource + subject. Cache invalidation on policy changes.
- **Sub-5ms p99** — Policy evaluation targets <5ms at the 99th percentile for cache hits, <20ms for full evaluation with content inspection.

The decision cache specifically excludes high-risk actions (risk score > 0.7) and actions with requirements—these always receive fresh evaluation. Cache metrics (hit rate, miss rate, eviction count) are exposed via the observability engine.

13. Semantic DLP: LLM-Powered Content Classification

While pattern-based DLP (Section 4) catches structured sensitive data (SSNs, credit cards), many data leaks involve unstructured content that does not match regex patterns. Semantic DLP classifies content by *meaning* rather than pattern.

The SemanticAnalyzer maintains a taxonomy of sensitive categories: trade secrets, legal privilege, medical records, financial projections, employee reviews, strategic plans, source code, authentication material, and customer data. Each category has configurable severity and response actions.

Classification can use local heuristics (keyword density, document structure analysis) or external LLM evaluation. Results feed into policy decisions: a trade secret classification triggers deny or redaction requirements even when no regex pattern matches. This closes the gap between what pattern-based DLP catches and what actually constitutes sensitive information.

14. Explainable Denials: Human-Readable Rejection Reasoning

When Antihero denies an action, the agent (and its human operator) deserves to know *why*. The ExplanationEngine generates multi-layer explanations for every denial:

- **Plain-language summary** — One sentence explaining the denial in non-technical terms
- **Technical detail** — Matched rule, policy tier, effect, risk score, and contributing factors
- **Remediation suggestions** — Concrete steps to resolve the denial (request approval, use a different resource, reduce scope)
- **Policy references** — Links to the specific policy rules that caused the denial

Explanations are attached to Policy Decision Envelopes and exposed via API. They support debugging agent behavior, training operators, and satisfying EU AI Act transparency requirements for high-risk system decisions.

15. Policy Simulation: Pre-Deployment Risk Testing

Before deploying policy changes to production, operators can simulate the impact against historical traffic. The PolicySimulator replays recorded AEEs against proposed policies and reports which decisions would change.

Simulation output includes: total events replayed, decisions changed, deny-to-allow changes (risk: false negatives), allow-to-deny changes (risk: operational disruption), and per-rule impact scores. This prevents “policy outages” where a new rule unexpectedly blocks critical agent workflows.

Simulation integrates with the Digital Twin engine (Section 26) for advanced what-if branching at the Sovereign tier.

16. Formal Verification: Mathematical Policy Correctness

Antihero’s formal verification engine proves mathematical properties about policy configurations. Three core properties are checked:

- **Completeness** — Every possible (action, resource) pair is covered by at least one policy rule. Gaps are flagged with the specific uncovered action/resource combinations.
- **Consistency** — No two rules at the same priority and tier produce contradictory effects for the same match. Contradictions are flagged with the conflicting rule pairs.

- **Deny dominance** — For every deny rule, no allow rule at any tier can override it. Violations are flagged with the specific allow rule that would override the deny.

Verification runs in sub-second time for typical policy sets (100–500 rules). Results include a proof certificate with rule coverage statistics, conflict pairs, and overall soundness determination. This is critical for government and regulated deployments where policy correctness must be provable, not just tested.

Part IV — Government-Ready Platform

Part IV describes ten capabilities designed for enterprise and government deployments. These features are gated behind subscription tiers to align pricing with deployment complexity. The following matrix summarizes tier access:

Feature	Enforcer	Sentinel	Sovereign
17. Policy Knowledge Graph	Basic graph	Full traversal	Custom ontology
18. Incident Response	—	Quarantine + escalation	Playbooks + evidence
19. Policy Version Control	Basic versions	Approval workflows	Branch/merge
20. Compliance Mapping	—	7 frameworks	Custom + reports
21. Multi-Tenant Hierarchy	—	2-level (org→team)	4-level hierarchy
22. Agent Observability	Basic metrics	Drift + alerts	SLA monitoring
23. FIPS Crypto & Air-Gap	—	—	Full FIPS + air-gap
24. Federated Policy Sync	—	—	Full federation
25. Policy Marketplace	Browse + install	Publish + rate	Private catalog
26. Digital Twin / Simulation	—	Basic replay	What-if branching

Table 1: Government-Ready Feature Tier Matrix

17. Policy Knowledge Graph

The PolicyKnowledgeGraph ingests policy documents and audit events to build a navigable graph of relationships between agents, policies, resources, actions, and rules. Five node types (Agent, Policy, Resource, Action, Rule) and six edge types (governs, permits, denies, accesses, triggers, contains) model the complete security topology.

Key queries include: **Agent Reachability** (what resources can an agent access?), **Resource Governance** (which policies protect a resource?), **Coverage Gaps** (which resources have no governing policy?), and **Impact Analysis** (if this rule is removed, what becomes unprotected?). The graph is

exportable as D3.js-compatible JSON for interactive visualization.

For government deployments, the knowledge graph provides instant answers to auditor questions: “Show me every agent that can access classified data” or “What policies govern database writes in the production environment?”

18. Incident Response Automation

The IncidentManager provides automated incident lifecycle management. When the Guard pipeline detects a high-severity threat (risk score ≥ 0.9) or trajectory match (severity ≥ 0.8), it automatically creates an incident.

Incidents support four severity levels (Low, Medium, High, Critical) and four states (Open, Investigating, Contained, Resolved). Three quarantine action types isolate threats: **disable_agent**, **block_resource**, and **freeze_session**. Quarantine checks are enforced at the top of the Guard pipeline—before any other evaluation—ensuring quarantined agents cannot execute any action.

Escalation chains define time-based notification sequences. Evidence bags preserve the complete context of an incident: related AEE chains, session history, and policy state at the time of the incident. At the Sovereign tier, custom playbooks define automated response sequences for specific incident types.

19. Policy Version Control

PolicyVersionControl provides git-like version management for policy documents. Every change creates a new PolicyVersion with a SHA-256 content hash, author, timestamp, change description, and parent version link.

Core operations: **commit** (create a new version), **diff** (compare two versions with added/removed/modified rule tracking), **rollback** (revert to a previous version), and **history** (list all versions with metadata).

At the Sentinel tier, approval workflows require designated reviewers to approve policy changes before activation. Approval requests track reviewer identity, approval/rejection status, and timestamps.

At the Sovereign tier, branch-and-merge operations enable parallel policy development. Teams can create feature branches, develop policies independently, and merge them with conflict detection and resolution. This mirrors the software development workflow that engineering teams already understand.

20. Compliance Mapping Engine

The ComplianceMappingEngine maps Antihero's technical controls to regulatory framework requirements. Seven built-in frameworks are included:

- **SOC 2 Type II** — Trust Services Criteria (security, availability, confidentiality)
- **HIPAA** — Technical and administrative safeguards for PHI
- **EU AI Act** — High-risk system requirements (Articles 9–15)
- **NIST AI RMF** — Govern, Map, Measure, Manage functions
- **NIST 800-53** — Federal information system security controls
- **FedRAMP** — Federal cloud security authorization
- **EO 14110** — Executive Order on Safe, Secure, and Trustworthy AI

For each framework, the engine performs automated **posture assessment** (what percentage of controls are satisfied by current configuration?), **gap analysis** (which controls are unsatisfied and what evidence is missing?), and **audit report generation** (formatted reports with control-by-control status and evidence references).

At the Sovereign tier, organizations can register **custom compliance frameworks** with their own controls and evidence requirements. This supports industry-specific regulations, internal policies, and contractual obligations.

21. Multi-Tenant Policy Hierarchy

The HierarchyEngine manages policy composition across a four-level organizational structure: **Organization** → **Team** → **Project** → **Agent**. Each level can define policies that inherit from and extend parent-level policies.

Composition semantics follow the same deny-dominates rule as the core policy engine: child scopes can add restrictions but never remove them. A team-level deny rule cannot be overridden by a project-level allow. Requirements merge via union across all applicable scopes.

The build_engine_for_scope() method resolves all applicable policies for a given scope, maps them to the existing four composition tiers (baseline, org, app, user), and creates a standard PolicyEngine. This design requires zero changes to the core PolicyEngine internals—the hierarchy sits above and feeds into it.

Sentinel tier: 2-level hierarchy (Organization → Team). Sovereign tier: full 4-level hierarchy with unlimited depth.

22. Agent Observability Platform

The ObservabilityEngine records per-agent metrics in rolling time windows: actions per minute, deny rate, average latency, risk score trends, and top resources/actions. This provides real-time visibility into agent behavior across the fleet.

Drift Detection

The DriftDetector compares current agent behavior against a baseline snapshot. Drift is detected across multiple dimensions: action rate changes, deny rate changes, latency deviations, and risk score trend shifts. Each drift metric includes a severity score (0–1.0) and the absolute change from baseline.

Alert Management

AlertRules define threshold-based triggers on any agent metric. Rules specify: metric name, comparison operator, threshold, evaluation window, cooldown period, and severity. When a rule fires, an AlertEvent is generated and routed through the notification system.

SLA Monitoring (Sovereign)

SLA targets define enforcement latency guarantees (e.g., “p99 ≤ 10ms”). The SLAMonitor continuously checks whether targets are being met and flags violations. This is critical for enterprise SLAs where enforcement latency must be provably bounded.

23. FIPS Cryptography and Air-Gap Deployment

For government and classified deployments, Antihero supports FIPS-mode cryptography and air-gapped operation.

FIPS Cryptographic Provider

The FIPSCryptoProvider uses Python’s hashlib with the OpenSSL backend (FIPS-capable when the system OpenSSL is FIPS-certified). Operations: SHA-256 hashing, HMAC-SHA-256 signing and verification. AEE chain signatures use HMAC-SHA-256 in FIPS mode, falling back to Ed25519 when FIPS is not required.

Classification Markers

Four classification levels are supported: UNCLASSIFIED (0), CUI (1), SECRET (2), TOP SECRET (3). Caveats (NOFORN, FVEY, etc.) can be attached. Classification banners are generated in standard format: “SECRET // NOFORN // FVEY”.

Air-Gap Mode

AirGapMode disables all network-dependent actions. When active, any TCE with a network-related action (http_request, api_call, send_email, etc.) is automatically denied before policy evaluation. This ensures that classified environments cannot accidentally leak data through agent actions.

Air-gap bundles enable secure data transfer between disconnected environments: export creates a signed, classified bundle; import validates signatures and classification markers before ingestion.

24. Federated Policy Sync

FederatedPolicySync enables policy exchange between organizations with different trust levels. Federation peers are registered with public keys and assigned one of three trust levels: **full_trust** (auto-merge), **review_required** (manual approval), and **read_only** (information only, no merging).

Policy bundles are cryptographically signed and contain metadata (author, timestamp, description) plus the policy documents themselves. The merge strategy follows Antihero’s core safety principle: **incoming deny rules always win**. Incoming allow rules only apply if the local policy set permits them. Requirements merge via union.

This enables multi-agency policy coordination (e.g., a federal department sharing baseline security policies with subordinate agencies) without compromising the safety properties of any individual deployment. Each organization retains full control of its local policy set while benefiting from shared policy intelligence.

25. Policy Marketplace

The Policy Marketplace provides a catalog of community and Antihero-curated policy templates for common use cases. Ten built-in templates cover major verticals:

- Healthcare (HIPAA) — PHI protection, minimum necessary access
- Finance (PCI DSS) — Payment card data isolation, audit logging

- Government (NIST 800-53) — Federal security controls, default-deny
- Education (FERPA) — Student record protection, age-appropriate content
- GDPR — Consent-based processing, cross-border transfer controls
- FedRAMP — Federal cloud security, continuous monitoring
- General security — API key protection, rate limiting, common patterns
- AI safety — Output filtering, hallucination guards, prompt injection defense
- DevOps — Production access controls, change management gates
- Enterprise — Data classification, DLP rules, executive action limits

Templates are installable with compatibility checking: the InstallManager validates that a template does not conflict with existing policies before installation. At the Sentinel tier, organizations can publish their own templates and rate community entries. Sovereign tier adds private marketplace instances.

26. Digital Twin and Simulation Environment

The DigitalTwin replays historical AEE chains against proposed policy configurations. This enables pre-deployment impact analysis: operators can answer “what would happen if we deployed this policy change last Tuesday?” without affecting production.

Simulation results include per-event comparisons (original effect vs. simulated effect), aggregate impact summaries (events changed, deny-to-allow count, allow-to-deny count), and change classification (safety improvement vs. operational disruption).

At the Sovereign tier, the WhatIfEngine supports **branching**: fork a simulation at any event, apply different policies to each branch, and compare outcomes. This enables A/B policy testing and “what-if” analysis for complex policy changes in government environments where production testing is not acceptable.

Part V — Formal Foundations

27. Deployment Surfaces

Antihero integrates at three surfaces:

Web

A browser extension wraps AI chat interfaces (ChatGPT, Claude, Gemini, etc.) with policy enforcement. Users see a security dashboard overlay showing real-time policy decisions, risk scores, and alert status. The extension communicates with the Antihero cloud API for policy evaluation and evidence recording.

Mobile

iOS and Android SDKs wrap agent tool calls with the same policy evaluation pipeline. Mobile-specific features include biometric MFA for high-risk confirmations and offline mode with air-gap bundle sync.

Local / Terminal

A daemon process wraps MCP servers, CLI tools, and IDE extensions. The MCP proxy intercepts tool calls transparently—agents don’t know they’re being governed. The IDE panel shows real-time policy decisions alongside code execution.

JavaScript / TypeScript SDK

The `@antihero/sdk` npm package provides a typed client for Node.js and browser environments. Zero runtime dependencies—uses native `fetch`. Every API endpoint is wrapped with full TypeScript interfaces matching the server-side Pydantic models: events, policies, claims, evaluation, billing, and dashboard statistics. Agent builders can integrate Antihero into any JavaScript agent framework with three lines of code.

28. Formal Properties

Antihero maintains four formal properties across all configurations:

P1: Fail-Closed. If the policy engine is unreachable, all actions are denied. No side effect executes without explicit evaluation.

P2: Deterministic. Given the same TCE and policy set, the policy engine always produces the same PDE. No randomness, no approximation, no "maybe."

P3: Monotonic Safety. Adding a deny rule to any tier can only reduce the set of allowed actions. No configuration change can expand permissions beyond what the base policy allows.

P4: Evidence Integrity. The AEE chain is append-only. No record can be modified or deleted without breaking the hash chain. Tampering is detectable.

These properties are verified by the Formal Verification engine (Section 16) and hold regardless of deployment surface, subscription tier, or feature configuration.

29. Limitations and Honest Boundaries

Antihero is transparent about what it does not protect against:

• **Model internals** — Antihero does not inspect model weights, activations, or training data. It operates at the action boundary, not the model boundary.

• **Side channels** — If an agent encodes information in the timing or ordering of allowed actions, Antihero's content inspection will not catch it (though behavioral analytics may flag unusual patterns).

• **Physical world effects** — Antihero governs digital actions (API calls, file operations, network requests). It cannot prevent physical-world consequences of allowed digital actions.

• **Policy quality** — Antihero enforces the policies you write. If policies are too permissive, agents will be too permissive. Formal verification and simulation help, but policy design remains a human responsibility.

• **100% attack prevention** — No security system prevents all attacks. Antihero dramatically reduces the attack surface and provides evidence for post-incident analysis, but residual risk always exists.

30. Conclusion

Antihero represents a fundamental shift in AI security: from trying to make models say the right things to ensuring they *do* the right things. By enforcing policy at the action boundary with cryptographic evidence and insurance-backed guarantees, Antihero provides the security infrastructure that the agentic AI era demands.

Version 3.0 extends this foundation with 17 new capabilities: seven advanced intelligence features (trajectory analysis, canary tokens, latency management, semantic DLP, explainable denials, policy simulation, formal verification) and ten government-ready platform features (knowledge graph, incident response, version control, compliance mapping, multi-tenant hierarchy, observability, FIPS crypto, federation, marketplace, digital twin).

The result is a platform that serves solo developers (Watchdog, free), growing teams (Enforcer, $29/mo), production deployments (Sentinel, $99/mo), and government agencies (Sovereign, custom pricing)—all running the same core engine with the same formal safety properties.

> *"Knowledge is action. Alignment at the action boundary is alignment where it matters."*

787 automated tests. Zero regressions. Fail-closed by default. Ship agents. Sleep at night.

Appendices

Appendix A: Formal Policy Algebra

Let P be the set of all policies, R the set of all rules, and E = {deny, allow, allow_with_requirements} the set of effects. Define the composition operator ⊕ as:

```
P■ ⊕ P■ := { r ∈ R■ ∪ R■ | if ∃ r′ ∈ R■ ∪ R■ with r′.effect = deny and r′.match ⊇ r.match, then r.effect := deny }
```

The deny-dominance property guarantees: for all tiers T■, T■, if any rule in T■ denies an action, no rule in T■ can override it. Requirements merge via set union: Req(P■ ⊕ P■) = Req(P■) ∪ Req(P■).

Appendix B: Cryptographic Specifications

AEE Hash Chain: Each AEE's hash = SHA-256(JCS(event) || previous_hash). Genesis event uses hash = SHA-256(JCS(event) || 0x00...00).

Ed25519 Signatures: Enterprise AEEs are signed with Ed25519 keys. Public keys are registered in the org's trust registry. Verification uses RFC 8032 pure Ed25519.

FIPS Mode: When enabled, all hash operations use hashlib.sha256() backed by OpenSSL FIPS provider. Signatures use HMAC-SHA256 instead of Ed25519.

JCS Canonicalization: RFC 8785 JSON Canonicalization Scheme ensures deterministic serialization regardless of key ordering or whitespace.

Appendix C: Insurance Economics

The dynamic risk pricing model uses six dimensions, each scored 0–1.0:

Dimension	Low Risk (0.0–0.3)	High Risk (0.7–1.0)
Capability Surface	Read-only tools	Shell exec, network, FS write
Autonomy Depth	Human-in-loop every action	Fully autonomous, no oversight
Network Openness	Air-gapped / internal only	Public internet, cross-org

Dimension	Low Risk (0.0–0.3)	High Risk (0.7–1.0)
Memory Persistence	Stateless / ephemeral	Persistent memory, fine-tuning
Human Gating	MFA + confirm on all actions	No human gates
Audit Integrity	FIPS signed, immutable chain	No audit, no signatures

Table 2: Risk Dimension Scoring

Composite risk score = weighted average of dimensions. Premium = base_rate × risk_multiplier × coverage_amount. Discounts for zero-claim periods and strong security posture. Surcharges for high deny rates and unresolved incidents.

Appendix D: Content Inspection Patterns

Built-in detection patterns (partial list):

Pattern	Type	Severity	Example
SSN	PII	Critical	123-45-6789
Credit Card	Financial	Critical	4111-1111-1111-1111
AWS Access Key	Secret	Critical	AKIA...
GitHub Token	Secret	High	ghp_...
JWT	Auth	High	eyJhbG...
Private Key (PEM)	Secret	Critical	-----BEGIN PRIVATE KEY-----
Email Address	PII	Medium	user@example.com
Phone Number	PII	Medium	+1-555-123-4567
IP Address	Infra	Low	192.168.1.1

Table 3: Content Inspection Patterns

Appendix E: Subscription Tier Feature Matrix

Complete feature availability by subscription tier:

Feature	Watchdog ($0)	Enforcer ($29)	Sentinel ($99)	Sovereign
Events / month	1,000	25,000	200,000	Unlimited
Policies	5	Unlimited	Unlimited	Unlimited
Audit retention	7 days	90 days	1 year	Unlimited
Policy engine	✓	✓	✓	✓
Content inspection	✓	✓	✓	✓
Behavioral analytics	✓	✓	✓	✓
AI insurance	—	—	$100K/incident	$1M+/incident
Trajectory analysis	—	✓	✓	✓
Canary tokens	—	✓	✓	✓
Decision caching	—	✓	✓	✓
Semantic DLP	—	—	✓	✓
Explainable denials	—	✓	✓	✓
Policy simulation	—	—	✓	✓
Formal verification	—	—	✓	✓
Knowledge graph	—	Basic	Full	Custom ontology
Incident response	—	—	✓	Custom playbooks
Version control	—	Basic	Approvals	Branch/merge
Compliance mapping	—	—	7 frameworks	Custom frameworks
Multi-tenant hierarchy	—	—	2-level	4-level
Observability	—	Basic metrics	Drift + alerts	SLA monitoring
FIPS crypto	—	—	—	✓
Air-gap deployment	—	—	—	✓
Federated policy sync	—	—	—	✓

Feature	Watchdog ($0)	Enforcer ($29)	Sentinel ($99)	Sovereign
Policy marketplace	—	Browse	Publish + rate	Private catalog
Digital twin	—	—	Replay	What-if branching
Support	Community	Email	Priority	Dedicated CSM

Table 4: Complete Subscription Tier Matrix

References: Bostrom (2014), Christian (2020), Tegmark (2017), Hinton (2023), EU AI Act (2024), NIST AI RMF (2023), NIST 800-53 Rev. 5 (2020), FedRAMP Authorization Act (2022), EO 14110 (2023), RFC 8785 (2020), RFC 8032 (2017).