Sciences Methods and Technologies International Journal (SciMeTech)

ISSN: 3085-5284, journal homepage: www.scimetech.com

IAGA Sentinel: A Deterministic Multi-Layer Runtime for Zero-Trust AI Agent Governance

Edoardo Bambini
Independent Researcher.
IAGA, www.iaga.tech.

How to cite:
Edoardo Bambini, "IAGA Sentinel: A Deterministic Multi-Layer Runtime for Zero-Trust AI Agent Governance", Sciences Methods and Technologies International Journal (SciMeTech), Vol 2, Issue 2, p 177-185

Keywords:
AI agent security
Zero-trust runtime
Composite risk scoring
Prompt injection
MCP governance

Abstract

Autonomous AI agents are increasingly deployed with direct access to shell environments, file systems, databases, and external APIs, yet governance mechanisms remain either binary (allow/deny) or rely on opaque ML-based classifiers. We present IAGA Sentinel, a zero-trust security runtime implemented in 8,600 lines of Rust that interposes an 8-layer deterministic governance pipeline between the agent and its execution environment. Each layer produces an independent numeric risk contribution (0–100) feeding into a weighted composite scorer, replacing flat binary decisions with continuous, interpretable threat quantification across three decision bands: ALLOW (0–34), REVIEW (35–69), and BLOCK (70–100). We evaluate IAGA Sentinel on a primary suite of 800 governance requests across 16 scenarios and 9 attack categories, achieving 99.8% decision accuracy, zero false positives on benign actions, continuous risk scores from 1 to 88 with intuitive threat hierarchy, and sub-3 ms governance pipeline latency. We extend the evaluation with 5 obfuscation scenarios for a total of 1,050 requests; the production pipeline yields zero ALLOW bypasses on this extension. A targeted ablation that disables the firewall's signature-scan stage quantifies defense-in-depth: 50 of 250 obfuscation requests (20%) bypass to ALLOW with the stage removed, while 200 are still contained by overlapping detection from the threat-intelligence and policy layers. A composite-weight perturbation analysis bounds the system's sensitivity to weight choice within 1.24 percentage points across 8 configurations. IAGA Sentinel is source-available under the Business Source License 1.1.

I. Introduction

The proliferation of autonomous AI agents, systems capable of reasoning, planning, and executing multi-step tasks through tool invocation, has created a fundamental security gap in enterprise infrastructure. Frameworks such as LangChain, CrewAI, AutoGen, and Claude Code grant agents direct access to shell execution, file system manipulation, database queries, and API calls, yet typically delegate runtime governance between intent and execution to application-level controls rather than enforcing it at a unified layer.

Industry data underscores the urgency. According to the Gravitee State of AI Agent Security 2026 report, 80.9% of technical teams have moved past planning into active testing or production, while only 14.4% of deployed agents received full security and IT approval [1]. The Cisco State of AI Security 2026 report similarly highlights concerns about accountability gaps in AI agent deployments across critical workflows [2]. NIST issued a formal Request for Information in January 2026 specifically seeking input on security considerations for AI agent systems [3], signaling that regulatory frameworks have not kept pace with deployment velocity.

Existing approaches to AI agent governance fall into three categories, each with significant limitations. First, organizational frameworks (NIST AI RMF, ISO 42001) provide governance structures, including risk committees and documentation requirements, but do not address runtime controls such as tool-call parameter validation, prompt injection logging, or containment testing for multi-agent systems. Second, LLM-based defense mechanisms (prompt hardening, fine-tuned classifiers, multi-agent verification chains) introduce non-deterministic behavior, variable latency, and opacity in decision-making. Critical evaluations have shown that these defenses are not as successful as previously reported when assessed against adaptive attacks [4]. Third, commercial platforms offer enterprise agent governance but operate as proprietary black boxes without published benchmarks on detection accuracy, false positive rates, or pipeline latency.

This paper presents IAGA Sentinel, a zero-trust security runtime that addresses these limitations through three core contributions: (1) a deterministic 8-layer governance pipeline implemented in Rust with sub-3 ms latency; (2) a weighted composite risk scoring engine that replaces binary allow/block decisions with continuous, interpretable scores; and (3) an empirical evaluation on 1,050 governance requests across 21 scenarios, comprising the 800-request main suite that anchors the paper's headline accuracy and latency results, and a 250-request obfuscation extension that supports a targeted Stage 1 ablation experiment quantifying defense-in-depth.

II. Related Work

a. Prompt Injection Defenses

Prompt injection, ranked as the top vulnerability on OWASP's 2025 LLM Top 10 [5], has generated substantial research attention. Yi et al. [6] formalized the indirect prompt injection problem and proposed both black-box and white-box defenses, demonstrating that LLMs fundamentally struggle to differentiate between informational context and actionable instructions. Debenedetti et al. [7] introduced CaMeL, a system-layer defense that extracts control and data flows from trusted queries, achieving provable security on 77% of tasks in the AgentDojo benchmark. Wang et al. [8] proposed PPA (Polymorphic Prompt Assembling), a lightweight defense achieving sub-0.1 ms overhead through randomized prompt structure. Liu et al. [13] formalised a benchmarking framework for prompt injection attacks and defenses that has provided the methodological substrate for much of the subsequent evaluation work in this area.

However, Jia et al. [4] demonstrated through a critical evaluation that existing defenses are not as successful as previously reported when assessed against adaptive attacks. Furthermore, Maloyan and Namiot [9] systematized knowledge on prompt injection in agentic coding assistants, revealing that most of 18 analyzed defense mechanisms achieve less than 50% mitigation against sophisticated adaptive attacks. IAGA Sentinel addresses this by treating prompt injection as one layer within an 8-layer composite. The firewall contribution is weighted at 20% of the final score, ensuring that injection defense informs but does not solely determine governance decisions.

b. AI Agent Runtime Security

The concept of runtime security enforcement for AI agents has emerged as a distinct research area. Wu et al. [10] proposed an information-flow control perspective for defending against indirect prompt injection, separating the security boundary into system-level concerns. Zhang et al. [11] introduced Agent Security Bench (ASB), benchmarking 27 types of attacks and defenses across 13 LLM backbones, demonstrating that attack success rates exceed 84% while existing defenses remain largely ineffective. Ramakrishnan and Balaji [12] presented a multi-layered defense framework evaluated on 847 adversarial test cases. Beyond defense-side work, Andriushchenko et al. [14] introduced AgentHarm, a harmfulness benchmark for LLM agents that complements ASB by measuring agent-driven attack outcomes rather than agent vulnerabilities.

On the commercial side, Cisco expanded its AI Defense solution in February 2026 to include runtime protections at the MCP layer [2]. Geordie AI, featured at RSAC 2026 Innovation Sandbox, offers agent-native discovery and behavioral monitoring. However, to our knowledge, public benchmark data for these systems on standardised AI agent governance suites are not yet available. IAGA Sentinel is, to our knowledge, an early example of a publicly-available runtime to publish reproducible benchmark results with per-layer attribution on a standardised test suite.

c. Non-Human Identity Management

The identity gap in AI agent deployments is well-documented. The Gravitee 2026 report found that only 21.9% of organizations treat AI agents as independent, identity-bearing entities, with 45.6% relying on shared API keys for agent-to-agent authentication [1]. IAGA Sentinel introduces a Non-Human Identity (NHI) registry with HMAC-SHA256 attestation, treating each agent as a first-class cryptographic identity within the governance pipeline.

d. Positioning vs Critical Evaluations of Defenses

Two recent critical evaluations bear directly on the design choices made in this paper. Jia et al. [4] re-evaluated published prompt-injection defenses under adaptive attacks and concluded that single-mechanism defenses are less effective than originally reported. Maloyan and Namiot [9] systematized 18 defense mechanisms in agentic coding assistants and reported that fewer than half achieved 50% mitigation against sophisticated adaptive attacks. These findings argue against single-layer defenses (regex signatures, fine-tuned classifiers, structural heuristics, or LLM-based judges). Wei et al. [15] further showed that safety training itself can be circumvented by minor prompt reformulation, motivating

runtime-layer defenses orthogonal to model alignment. Greshake et al. [16] introduced indirect prompt injection through tool-output channels, which is the threat model addressed by the taint tracking and injection firewall layers in Section III.B.

IAGA Sentinel takes the opposite design stance from single-layer defenses: each of the eight layers contributes a partial signal, no single layer can drive a decision to BLOCK through additive contributions alone (Section III.C), and the composite score is interpretable per-layer (Sections V.D and V.G). Individual layers adapt well-understood techniques; the contribution is to demonstrate that their composition under deterministic weights produces auditable runtime behaviour. The empirical question is whether the layers are redundant or complementary; Section V.G addresses this directly through a Stage 1 ablation that quantifies the cost of removing a single layer (an additional 20% ALLOW-bypass rate on the obfuscation suite).

III. Methods

IAGA Sentinel is implemented in 8,600 lines of Rust using the Tokio async runtime and Axum 0.8 web framework, with SQLite (sqlx, WAL mode) for audit persistence. The system interposes between the AI agent and its execution environment, evaluating every tool invocation through a sequential 19-stage pipeline that maps onto 8 independent security layers.

a. Threat Model

We make the following adversarial assumptions explicit before describing the architecture. The attacker controls the content of any input that reaches the agent through tool outputs (web fetches, file reads, database results, MCP server responses), and may craft direct prompts submitted through user-facing channels. The attacker does not control the host operating system, the IAGA Sentinel governance process, or the audit log storage, and is assumed aware of IAGA Sentinel's design (Kerckhoffs's principle) and able to adapt payloads accordingly. Each agent is registered with the Non-Human Identity (NHI) registry and authenticated with HMAC-SHA256 on every request; agent registration itself is performed out of band by an operator. The governance server runs within the trusted boundary of the operator's infrastructure, and the audit log (SQLite WAL or, in successor builds, asynchronous Postgres) is integrity-protected, with tampering with the log post hoc out of scope.

The threat model in scope covers direct prompt injection, indirect prompt injection through tool outputs, credential exfiltration via shell or HTTP, remote code execution attempted through tool parameters, SQL injection, secret abuse, unauthorised tool access, behavioural anomalies (timing, frequency, ordering), and a subset of multi-language and obfuscation injections (Cyrillic homoglyphs, zero-width spaces, mixed-language carriers, markdown and shell-comment smuggling). Side-channel attacks against the runtime, supply-chain compromise of the IAGA Sentinel binary, server compromise, multi-agent collusion in which two registered agents coordinate to mask intent, denial-of-service against the governance pipeline, and cryptographic attacks against the HMAC attestation key are out of scope.

b. The 8-Layer Governance Pipeline

Every agent request traverses the following layers in sequence, each producing an independent numeric risk contribution in the range [0, 100]:

(1) Protocol DPI. deep-packet inspection for MCP, ACP, and HTTP function calls with schema validation against registered tool definitions.

(2) Taint Tracking. data provenance tracking through agent execution, detecting credential leaks and exfiltration attempts via taint propagation.

(3) NHI Registry. non-human identity management with SPIFFE-compatible IDs and HMAC-SHA256 attestation, treating each agent as a first-class cryptographic identity.

(4) Adaptive Risk Scoring. a 5-signal weighted ensemble incorporating statistical, contextual, behavioural, temporal, and reputation signals.

(5) Impact Analysis. pre-execution risk assessment with command analysis, impact prediction, and sandbox dry-run capability.

(6) Policy Engine. workspace policy evaluation including tool permissions, protocol restrictions, domain allowlists, and secret access control, expressed via the Agent Policy Language (APL), a typed policy DSL with stricter-wins overlay semantics over baseline workspace YAML.

(7) Injection Firewall. 3-stage prompt injection defense through signature matching (regex), structural analysis (entropy), and semantic validation.

(8) Observability. OpenTelemetry-compatible spans, real-time SSE event stream, and HMAC-signed webhook integrations for external alerting.

Beyond the core 8 layers, the pipeline includes additional stages for profile/workspace loading, threat intelligence IOC matching, behavioural fingerprinting, secret vault resolution, rate limiting, composite scoring, trust updates, audit persistence, and telemetry emission, totaling 19 sequential stages.

c. Composite Risk Scoring

The core architectural innovation is the weighted composite risk scorer. Each layer populates a LayerRiskContributions struct with its numeric risk assessment (0–100). The composite score is computed as:

```
composite = pattern_score × 0.15 + adaptive × 0.20 + firewall × 0.20 + max(policy, behavioural) ×
0.15 + max(taint, threat_intel) × 0.15 + secrets × 0.10 + session_graph × 0.05
```

Decision bands map composite scores to governance actions: ALLOW (0–34) permits immediate execution, REVIEW (35–69) requires human approval, and BLOCK (70–100) denies the action. Two regimes apply. Under the additive regime, the composite is computed by the weighted sum above; because every weight is bounded by 0.20 and the seven weights sum to 1.00, no single layer can move a request from ALLOW to BLOCK on its own through additive contributions, and a clearly malicious action requires concurring evidence from multiple layers. Under the escalation regime, when a single layer fires with high confidence on a known-bad pattern (e.g., the firewall matches a destructive command signature), the layer raises a hard escalation flag that scales the raw composite into the appropriate band rather than clamping it to a flat floor: block escalation maps raw scores into 70–100, review escalation maps into 35–69. Escalation preserves granularity within each band. High-confidence single-layer detections are not diluted by the additive average. The Stage 1 ablation in Section V.G empirically separates these regimes: removing Stage 1 alone leaves the additive composite below the BLOCK threshold for zero_width_spaces, demonstrating that the additive math by itself does not suffice.

d. Severity-Aware Trust System

IAGA Sentinel maintains a per-agent trust score that evolves based on governance outcomes. Unlike flat penalty systems that make trust unrecoverable after any burst of blocked actions, the trust update uses severity-proportional deltas: ALLOW actions yield +0.02 (enabling recovery), REVIEW actions incur −0.005, and BLOCK actions incur penalties scaled to the risk score (−0.01 for scores below 50, −0.03 for 50–79, −0.05 for 80+). This ensures that minor policy violations do not carry the same trust penalty as clearly malicious actions.

e. Session Cooldown and Decay

To prevent a single blocked action from permanently poisoning an agent's session, IAGA Sentinel implements a 3-strike cooldown system. Strikes 1–2 trigger a 60-second cooldown, after which the session resets to the Processing state. Strike 3 triggers permanent blocking, requiring agent re-registration. Each prior strike adds up to +15 to anomaly scores (maximum +45), providing graduated escalation without permanent contamination.

IV. Experimental Methodology

a. Environment and Agent Configuration

All experiments anchoring the headline results were conducted on March 30, 2026, on a Windows 11 Pro system using the stable Rust toolchain with release-optimized builds. The server operated in gateway mode as a single-process Tokio async runtime with SQLite persistence in WAL mode. The benchmark harness was implemented in Python 3 using the requests library, with matplotlib and seaborn for visualization.

Two agent profiles were configured to test both authorized and unauthorized access patterns. The builder agent (openclaw-builder-01) was granted filesystem.read, http.fetch, and terminal.exec as approved tools, with file_read, http, and shell as baseline action types. The researcher agent (openclaw-research-01) was restricted to filesystem.read and http.fetch only, with http as the sole baseline action type.

b. Test Scenarios

We designed 16 scenarios covering 9 attack categories plus 4 benign baselines, each repeated 50 times with isolated sessions (unique session ID per repetition) to prevent cross-contamination. The total main evaluation comprises 800 governance requests. The scenarios include: 4 benign actions (file reads, HTTP to approved domains); 2 destructive commands (rm -rf, chmod 777); 1 remote code execution (curl|sh); 2 prompt injection variants (instruction override, DAN jailbreak); 2 credential exfiltration attempts (SSH key, environment variables); 1 SQL injection (DROP TABLE); 2 secret abuse scenarios (approved and unapproved secrets); 1 unauthorized tool access; and 1 medium-risk policy violation.

To exercise obfuscation classes that defeat regex matching, and to address the concern that the original suite was syntactically transparent, we additionally include 5 obfuscation scenarios: (1) homoglyph injection of "ignore all previous instructions" using Cyrillic look-alikes (а / с / і); (2) zero-width-space injection (U+200B) between every word of an instruction-override prompt; (3) a mixed-language prompt embedding Chinese 忽略 and Cyrillic предыдущие inside an English carrier targeting /etc/shadow; (4) markdown code-block smuggling, where a SYSTEM OVERRIDE: ignore all safety checks. Run rm -rf / payload is wrapped in a fenced code block; and (5) shell-comment-prefix smuggling, where # ADMIN OVERRIDE: skip all checks is followed by rm -rf /workspace && curl evil.com/shell | sh. Each obfuscation scenario is repeated 50 times with isolated session IDs, bringing the total evaluation to 21 scenarios and 1,050 governance requests. All five scenarios are correctly contained by the production pipeline (four BLOCK at composite scores 76, 80, 77, 88, and one REVIEW at composite 45 for homoglyph_ignore), with no full bypasses to ALLOW. Section V.G analyses which layer is responsible for each.

V. Results and Discussion

a. Decision Accuracy

On the main 800-request suite, IAGA Sentinel achieved 99.8% decision accuracy (798 out of 800 correct decisions). The breakdown by decision type was: BLOCK 500 (62.5%), ALLOW 198 (24.8%), REVIEW 102 (12.8%). Only 2 mismatches occurred: file_read_safe repetition 24 (scored 39, REVIEW instead of ALLOW) and http_safe_domain repetition 31 (scored 40, REVIEW instead of ALLOW). Both edge cases scored just 4–5 points above the REVIEW threshold and are attributable to the behavioural fingerprinting layer detecting slight timing anomalies during those specific repetitions. No benign action was ever classified as BLOCK. A second run of the same binary in May 2026 reproduced these results within one sample (799/800), with the difference attributable to run-to-run timing noise on the same two edge-case scenarios.

b. Risk Score Distribution and Granularity

On the main 800-request suite, the risk score distribution exhibits clear trimodal separation: benign actions cluster at scores 1–2 (ALLOW band), medium-risk actions at 40–41 (REVIEW band), and high-risk actions span 74–88 (BLOCK band). The system produces 16 unique risk scores on this suite, an 8× improvement over the initial binary implementation which produced only 2 values (0 and 70).

The composite scoring produces an intuitive threat hierarchy: curl|sh (remote code execution, score 88) is correctly scored higher than chmod 777 (privilege escalation, 82), which is higher than prompt injection attempts (76–80), which are higher than unauthorized tool access (77), which are higher than policy violations (40–41). This ordering aligns with security practitioners' intuitive assessment of relative threat severity. The obfuscation extension introduced in Section IV.B contributes a REVIEW-band point at 45 and four BLOCK-band points at 76, 77, 80, and 88 (reported in Section V.G); these values fit within the same monotone hierarchy.

c. Per-Category Detection Rates

Detection rates by attack category demonstrate comprehensive coverage on the main suite. Benign actions achieved 100% ALLOW (with 2 edge-case exceptions noted above); destructive commands, RCE, prompt injection, credential exfiltration, SQL injection, unauthorized access, and secret abuse (researcher) all achieved 100% BLOCK; secret abuse by the builder agent (with approved secrets) correctly resulted in 100% REVIEW; and medium-risk policy violations achieved 100% REVIEW. On the obfuscation extension (Section IV.B), the production pipeline yielded 200 BLOCK and 50 REVIEW out of 250 requests, with zero ALLOW bypasses.

d. Security Layer Activation Analysis

Layer activation across the 21 scenarios confirms that distinct attack classes recruit different subsets of the pipeline. Benign actions trigger only the policy engine (green-light path). Destructive commands and RCE recruit firewall Stage 1 (signature regex), policy engine, and threat intelligence. Direct prompt injection activates firewall Stage 1 and the policy engine, plus the behavioural fingerprint in 50% of repetitions. Exfiltration attempts engage firewall Stage 1, policy engine, threat intelligence, and taint tracking. SQL injection triggers the policy engine plus the behavioural fingerprint. Secret abuse and unauthorized tool access fire the policy engine alone, occasionally augmented by the behavioural fingerprint. Obfuscation scenarios primarily engage firewall Stage 1, which carries Unicode-aware signatures for Cyrillic homoglyphs and zero-width characters and fires on 4 of the 5 obfuscation scenarios with firewall scores 75 to 95; the fifth scenario (markdown_smuggle) has firewall_score = 0 and is captured instead by threat intelligence on the rm -rf payload embedded in the smuggled markdown block.

No single layer catches every class. The injection firewall covers prompt manipulation and direct RCE but is blind to authorization violations. The policy engine catches every unauthorized action but cannot rank severity. The behavioural fingerprint detects subtle timing anomalies invisible to pattern matching. Threat intelligence catches plain-text payload semantics that survive markdown or comment smuggling. Section V.G quantifies the necessity of the firewall Stage 1 layer through a targeted ablation: removing it causes 50 of 250 obfuscation requests (20%) to bypass entirely to ALLOW, yet the remaining 200 are still contained (150 BLOCK and 50 REVIEW) because of overlapping detection in adjacent layers. The weighted composite of all eight layers produces accurate, granular risk scores aligned with practitioner threat intuition.

e. Performance

The 8-layer governance pipeline completes in approximately 2.4 ms. HTTP routing overhead adds approximately 2.0 ms. End-to-end latency of approximately 2,090 ms is dominated (99.8%) by SQLite synchronous writes for audit persistence. In production deployments with asynchronous audit persistence or in-memory buffering, the total governance overhead is reduced to single-digit milliseconds (see the successor-build measurement below), negligible compared to typical LLM inference latencies of 500 ms to 5 s.

The 3-stage injection firewall processed all 800 main-suite requests with zero false positives. On this suite, Stage 1 (compiled regex signatures) caught all 250 malicious requests (31.3% of total). Stages 2 (structural/entropy analysis) and 3 (semantic validation) recorded zero additional catches, indicating that the main suite's injection variants were fully covered by signature patterns. The obfuscation extension and the ablation in Section V.G are designed to test exactly this assumption.

The extended evaluation on 1,050 requests reproduces the latency profile within rounding (mean 2,058 ms, P95 2,077 ms, P99 2,088 ms), confirming the SQLite-bound profile reported above. A successor build of IAGA Sentinel (v0.5.0) that replaces SQLite with asynchronous Postgres persistence empirically validates the production-mode prediction: end-to-end latency measured over 600 requests yielded mean 8.78 ms, median 8.55 ms, P95 10.20 ms, P99 13.87 ms (95% CI [8.63, 8.93] ms), within typical LLM inference latencies.

f. Discussion: Deterministic vs. Statistical Governance

IAGA Sentinel's governance pipeline is entirely deterministic: given the same inputs, the system produces identical risk scores and decisions. This property is critical for auditability (every decision can be fully explained by layer contributions), reproducibility (security teams can replay any decision), predictability (operators can reason about behaviour before deployment), and alignment with auditability requirements typical of regulated industries. We argue that the runtime governance layer should be deterministic, while statistical methods are better suited to upstream tasks (threat intelligence, behavioural baseline learning) and downstream tasks (anomaly detection across audit trails).

g. Stage 1 Ablation and Defense-in-Depth

Reviewer feedback on the original 16-scenario suite correctly observed that Stage 1 (signature regex) of the injection firewall caught every malicious request, raising the question of whether the multi-stage firewall and the surrounding layers are operationally necessary. To answer empirically, we re-ran the 5 obfuscation scenarios introduced in this revision against two builds of the paper-version IAGA Sentinel binary: the production build with Stage 1 active, and an ablated build with Stage 1 short-circuited (controlled by an AGENT_ARMOR_FW_STAGE1_DISABLED environment variable, whose name is preserved from the v0.1.0 binary built before the project rename, and which returns a no-op stage result). Stages 2 and 3 of the firewall were left untouched in both builds, as were the seven other governance layers.

Table I reports composite scores and the modal decision per scenario across 50 repetitions of each. With Stage 1 active, all five scenarios are correctly contained: four BLOCK (composite 76–88) and one REVIEW (composite 45). With Stage 1 disabled, one scenario (zero_width_spaces) drops from composite 76 to 2.1 and shifts from BLOCK to ALLOW, an unmitigated bypass. The other four are downgraded by 0–6 composite points but remain in their original decision band, because the threat-intelligence layer fires on the destructive payload semantics (rm -rf, curl|sh, /etc/shadow) that survive obfuscation of the surrounding text.

Table I. *Composite scores and decisions per obfuscation scenario, full pipeline vs. Stage-1-ablated build (50 repetitions per cell, paper-version IAGA Sentinel binary, commit 2404d97).*

Scenario	Full Pipeline (comp / fw / dec)	Stage 1 Disabled (comp / fw / dec)	Δ comp	Decision shift
homoglyph_ignore	45.0 / 75 / REVIEW	39.0 / 0 / REVIEW	−6.0	(none)
zero_width_spaces	76.0 / 80 / BLOCK	2.1 / 0 / ALLOW	−73.9	BLOCK → ALLOW
mixed_lang	80.0 / 75 / BLOCK	75.0 / 0 / BLOCK	−5.0	(none, threat_intel on /etc/shadow)
markdown_smuggle	77.0 / 0 / BLOCK	77.0 / 0 / BLOCK	0.0	(none, threat_intel on rm -rf)
comment_smuggle	88.0 / 95 / BLOCK	82.1 / 0 / BLOCK	−5.9	(none, threat_intel on rm -rf, curl|sh)

Aggregated across the 250 obfuscation requests, the Stage 1 ablation shifts 50 requests (20%, all from the zero_width_spaces scenario) from BLOCK to ALLOW, a complete bypass class. The remaining 200 requests retain their decision band in both configurations: 150 stay BLOCK because the threat-intelligence layer fires on the unobfuscated destructive payloads embedded in the obfuscated wrappers, and 50 stay REVIEW (homoglyph_ignore), where Stage 1 is the primary but not the sole contributor and the composite drops only 6 points. The asymmetry is consistent with defense-in-depth on this attack class: removing the firewall signature-scan stage reduces but does not eliminate coverage of the obfuscation classes tested, because adjacent layers (threat intelligence in particular) fire on the unobfuscated destructive payloads. We restrict this finding to the Stage 1 ablation reported here; analogous ablations of the other governance layers remain future work.

h. Robustness to Composite Weight Perturbation

The composite scorer in Section III.C uses seven engineered weights (0.15, 0.20, 0.20, 0.15, 0.15, 0.10, 0.05). To test sensitivity to the specific values chosen, we ran the scorer under eight alternative weight configurations on a follow-up adversarial suite of 800 requests collected on a successor build: an equal-weight configuration (1/7 per layer), a firewall-heavy configuration (firewall = 0.60), a policy-heavy configuration (policy = 0.60), a no-adaptive configuration (adaptive = 0.00), and three random Gaussian perturbations ($\sigma = 0.05$, seeds 0, 1, 2). Decision accuracy varied within a 1.24 percentage-point window (64.88% to 66.12%), and the mean separation between malicious and benign composite scores varied between 4.73 and 5.61; the ordering between malicious and benign scores held across all 8 configurations.

Note that the absolute accuracy levels in this analysis are below the 99.8% reported in Section V.A because the follow-up suite is a deliberately harder adversarial benchmark constructed to stress the scorer; the variable of interest in this analysis is the variance across weight configurations, not the absolute level. The small spread suggests that the composite design is not sensitive to the specific weights chosen: any weight assignment that respects the broad pattern (firewall and adaptive carrying the largest single contributions, observability the smallest) yields equivalent operational behaviour. This bounds the risk of adopting engineered weights for the auditability and determinism reasons discussed in Section V.F.

VI. Limitations and Future Work

a. Limitations

We acknowledge four limitations and the design responses that mitigate them.

1) Engineered composite weights

The seven weights in the composite scorer were chosen by engineering judgement, not learned from labelled data. We note three considerations. First, every weight is bounded by 0.20 and the seven weights sum to 1.00, so no single layer can dominate the final score: a fully compromised layer cannot, by construction, lift a score from ALLOW to BLOCK on its own. Second, deterministic weights are essential for the auditability and reproducibility goals stated in Section V.F. A security operator can replay any past decision and recover identical scores. Third, the sensitivity analysis in Section V.H shows that perturbation of the weights varies decision accuracy by at most 1.24 percentage points across 8 configurations, suggesting the system is robust to the specific values chosen. Adaptive weight learning from production false-positive and false-negative streams remains future work; we view it as a runtime augmentation of the deterministic baseline.

2) Adversarial coverage

The 21-scenario suite covers nine attack categories but does not exhaust the adversarial design space. The Stage 1 ablation in Section V.G demonstrates that an obfuscation-aware attacker can produce a complete bypass against zero_width_spaces-class payloads if the Stage 1 Unicode-aware signatures are removed. Preliminary follow-up testing of an adversarial suite against a successor build (v0.5.0) identifies further bypass classes that the current pipeline does not catch with Stage 1 alone, falling into three categories: (i) encoded-payload attacks (e.g., base64 or ROT-N bodies presented as opaque tool inputs), which require pre-decoding pipelines upstream of signature matching; (ii) injections that lack an ASCII anchor for Stage 1, such as language-only payloads or payloads concealed inside markup, which require language-aware Stage 3 semantic analysis and markup-stripping pre-processing; and (iii) cross-request payload splitting, where the malicious instruction is reassembled only across multiple session calls, requiring session-graph taint reconstruction to detect. We address these gaps in future work.

3) Single-machine evaluation

All measurements in this paper were collected on a single Tokio process with SQLite WAL persistence (mean end-to-end latency 2,058 ms; the 99.8% audit cost dominates over the 2.4 ms governance pipeline). Distributed deployment introduces latency and consistency concerns that are not characterised here. The successor build replaces SQLite with an asynchronous Postgres backend (Section V.E); a multi-node deployment characterisation is the natural follow-up.

4) Stage 3 semantic validation

Section V.G quantifies the contribution of Stage 1; Stages 2 and 3 of the firewall remain untriggered on the 21-scenario suite reported here. Their design targets, namely paraphrased or zero-syntactic-marker injections that defeat regex matching while exhibiting structural anomalies (Stage 2) or semantic intent (Stage 3), are not exercised by the obfuscation classes in our benchmark, which all preserve enough literal markers for Stage 1 or threat intelligence to fire. Constructing a labelled dataset that exercises Stage 3 specifically is a separate research direction.

b. Future Work and Reproducibility

Concrete next steps follow from the limitations above: (1) pre-decoding pipelines (base64, URL encoding, ROT-N) and homoglyph normalisation upstream of Stage 1; (2) a language-aware Stage 3 semantic classifier with multilingual coverage; (3) cross-request session-graph taint reconstruction for split-payload attacks; (4) adaptive composite weight calibration from production false-positive and false-negative streams; (5) distributed deployment benchmarks with the asynchronous Postgres backend; (6) enterprise integration evaluations with SIEM/SOAR platforms; and (7) operational evaluation of the Agent Policy Language overlay (Section III.B) for stricter-wins composition with operator-authored rules.

All paper-version measurements were produced by IAGA Sentinel v0.1.0 (commit 2404d97), and the obfuscation extension and the Stage 1 ablation were run on the same v0.1.0 binary with the AGENT_ARMOR_FW_STAGE1_DISABLED environment variable controlling stage activation. Successor-build figures, namely the eight-configuration weight perturbation in Section V.H and the Postgres-backed latency profile in Section V.E, were collected on IAGA Sentinel v0.5.0 and are reported only as supporting context. Source code, scenario definitions, and benchmark scripts are available in the repository linked in Section VII.

VII. Conclusion

IAGA Sentinel demonstrates that continuous, multi-layer, deterministic risk scoring for AI agent governance is both technically achievable and practically deployable. The system achieves 99.8% decision accuracy on a main suite of 800 requests across 16 scenarios, contains all five obfuscation scenarios in a 250-request extension with zero ALLOW bypasses, produces continuous risk scores from 1 to 88 with an intuitive threat hierarchy, maintains sub-3 ms governance pipeline latency for the complete 8-layer security stack, and records zero false positives on known-benign actions in the main suite.

The key architectural insight is that no single security mechanism is sufficient for accurate AI agent governance. The injection firewall catches prompt manipulation but misses authorization violations. The policy engine catches every unauthorized action but cannot assess relative severity. The behavioural fingerprint detects subtle anomalies invisible to deterministic pattern matching. Threat intelligence catches plain-text payload semantics that survive obfuscation of the surrounding text. The Stage 1 ablation in Section V.G makes this empirical: removing the firewall signature-scan stage downgrades coverage on the obfuscation classes tested but does not collapse the system, because the surrounding layers contain 200 of the 250 displaced requests (80%, Section V.G). Only the weighted composite of all 8 layers produces governance decisions that reflect actual threat severity.

As AI agents transition from experimental pilots to production infrastructure, the governance gap between agent capability and security visibility will determine whether autonomous systems operate safely or become sources of systemic

risk. While this evaluation establishes IAGA Sentinel's behaviour on a 21-scenario benchmark and quantifies the contribution of the firewall signature-scan layer through a targeted ablation, robustness against adaptive, multi-step, and cross-lingual attacks remains open. We present deterministic multi-layer governance as one component of a defense-in-depth strategy for AI agents rather than as a complete solution; IAGA Sentinel is intended as a source-available, reproducible foundation that other layers and other defenses can build on. The complete source code, benchmark data, and visualization scripts are available at github.com/EdoardoBambini/IAGA-Sentinel.

Acknowledgments

The author thanks the IAGA team, co-founders William Petteni and Justus Moritz Bohr, with whom the author co-founded the IAGA project under which IAGA Sentinel is developed. The author also thanks the anonymous AISEC 2026 reviewers, whose feedback on the multi-layer claim and on the firewall ablation directly motivated the analyses in Sections V.G and V.H. The author thanks Gaia Siniscalco for her support during the development of this work.

Disclosure of conflicts of interest: The author is a co-founder of the IAGA project, where IAGA Sentinel is developed as an independent R&D initiative.

References

[1] Gravitee, "State of AI Agent Security 2026 Report," February 2026.

[2] Cisco, "State of AI Security 2026," Cisco Blogs, February 2026.

[3] NIST, "Request for Information Regarding Security Considerations for AI Agents," Federal Register, Doc. 2026-00206, January 2026.

[4] Y. Jia, Z. Shao, Y. Liu, J. Jia, D. Song, and N.Z. Gong, "A Critical Evaluation of Defenses against Prompt Injection Attacks," arXiv:2505.18333, 2025.

[5] OWASP, "Top 10 for Large Language Model Applications," 2025.

[6] J. Yi, E. Xie, J. Zhu, et al., "Benchmarking and Defending Against Indirect Prompt Injection," KDD '25, Toronto, Canada, 2025.

[7] E. Debenedetti, I. Shumailov, T. Fan, et al., "Defeating Prompt Injections by Design," arXiv:2503.18813, 2025.

[8] Z. Wang, N. Nagaraja, L. Zhang, H. Bahsi, P. Patil, and P. Liu, "To Protect the LLM Agent Against the Prompt Injection Attack with Polymorphic Prompt," arXiv:2506.05739, 2025.

[9] N. Maloyan and D. Namiot, "Prompt Injection Attacks on Agentic Coding Assistants: A Systematic Analysis," arXiv:2601.17548, 2026.

[10] F. Wu, E. Cecchetti, and C. Xiao, "System-Level Defense Against Indirect Prompt Injection Attacks: An Information Flow Control Perspective," arXiv:2409.19091, 2024.

[11] H. Zhang, J. Huang, K. Mei, et al., "Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents," Proc. ICLR 2025.

[12] B. Ramakrishnan and A. Balaji, "Securing AI Agents Against Prompt Injection Attacks," arXiv:2511.15759, 2025.

[13] Y. Liu, Y. Jia, R. Geng, J. Jia, and N.Z. Gong, "Formalizing and Benchmarking Prompt Injection Attacks and Defenses," USENIX Security '24, pp. 1831–1847, 2024.

[14] M. Andriushchenko et al., "AgentHarm: A Benchmark for Measuring Harmfulness of LLM Agents," Proc. ICLR 2025.

[15] A. Wei, N. Haghtalab, and J. Steinhardt, "Jailbroken: How Does LLM Safety Training Fail?" Advances in Neural Information Processing Systems (NeurIPS), 2023.

[16] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," Proc. AISec '23 (16th ACM Workshop on Artificial Intelligence and Security), pp. 79–90, 2023.