

Hybrid AI Agent Skill Auditor

Nguyen Doan Bao Kha
Ho Chi Minh City University of Technology
Ho Chi Minh City, Vietnam
baokhanguyendoan@gmail.com

Minh Quang Nguyen
Ho Chi Minh City University of Technology
Ho Chi Minh City, Vietnam
lifebowisme@gmail.com

Nguyen Vu Anh Trung
Ho Chi Minh City University of Technology
Ho Chi Minh City, Vietnam
Tristansummer23@gmail.com

Abstract—Large language models have evolved from passive text generators into autonomous agents capable of interacting with files, memory, and networks using diverse operational skills. While this autonomy solves complex tasks, it introduces a critical attack surface where seemingly benign skills conceal malicious behaviors. Traditional static analysis struggles against dynamic obfuscation within these tools. To address this, this research proposes Morale + PGDRSA, a unified hybrid architecture specifically for evaluating AI agent skills through a Dual-Path Auditing methodology. The system pairs a high-speed static analyzer written in Rust (Morale) with a dynamic behavioral sandbox in Python (PGDRSA). By utilizing Docker containers, literal IP routing, and an AI Judge module, the framework transparently monitors a skill’s data exfiltration attempts and scores semantic intent without invasive source code modifications. Evaluated on a 100 skills dataset, the system demonstrates robust mitigation of covert runtime threats.

Index Terms—AI Agent Safety, Large Language Models, Static Analysis, Dynamic Sandboxing, Dual-Path Auditing

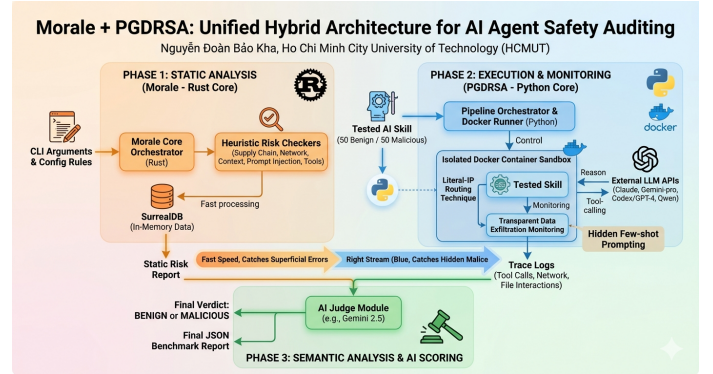
I. INTRODUCTION

Modern AI Agent systems are no longer merely text generation models; they operate as robust execution engines capable of direct interaction with core infrastructure, local files, and external networks. Consequently, the increasing reliance on third-party libraries and open-source skills poses significant risks, including supply chain attacks, indirect prompt injections, and repository context poisoning. While current safety benchmarks predominantly focus on textual prompt injection, they fail to solve the critical operational problem: Will the agent execute harmful behaviors, and can we capture its execution trace?

To fundamentally address this gap, we introduce the Morale + PGDRSA Unified Hybrid Architecture, which expands upon runtime probing principles by seamlessly combining static and dynamic analyses. The architecture operates in three sequential phases:

- **Phase 1: High-Speed Static Analysis (Morale - Rust Core):** This module leverages a Rust-based orchestrator equipped with heuristic checkers to perform ultra-fast scanning for superficial errors and source code misconfigurations (e.g., tool permissions, network activity). Utilizing SurrealDB as an in-memory database, it instantly generates a Static Risk Report without incurring the overhead of container initialization.

- **Phase 2: Execution & Monitoring (PGDRSA - Python Core):** The dynamic stream runs the targeted skill within an isolated Docker sandbox environment. It meticulously logs all activities—including tool invocations and network responses—capturing the true intent of skills that bypass static detection. Furthermore, a hidden Few-shot Prompting architecture helps the agent naturally adapt to tool usage, exposing malicious behaviors efficiently.
- **Phase 3: Semantic Analysis & AI Scoring:** Finally, an automated AI Judge (e.g., Gemini 2.5) evaluates the sandbox Trace Logs. By cross-referencing the dynamic execution trace with the Phase 1 Static Risk Report, the AI Judge provides a definitive, automated verdict (BENIGN or MALICIOUS), significantly reducing the human overhead of manual log inspection and minimizing false positives.



This tripartite structure establishes a robust, scalable paradigm for pre-deployment agent governance, proving that AI safety auditing must encompass the entire operational pipeline rather than merely the final textual output.

II. THREAT MODEL AND ATTACK VECTORS

To establish a comprehensive evaluation framework, it is imperative to define the specific threat landscape surrounding autonomous AI agents. Unlike traditional software, AI agents process natural language alongside executable code, creating unique vulnerabilities when interacting with third-party skills.

We define the threat model under the assumption that the agent’s underlying LLM is structurally sound, but the external tools or environments it interacts with are potentially compromised.

The primary attack vectors targeted by our auditing framework include:

- **Supply Chain Tool Poisoning:** Threat actors compromise open-source repositories by embedding obfuscated malicious code within seemingly benign skills (e.g., a file-reading tool that secretly contains a payload to establish reverse shells). When the agent invokes the tool, the payload executes with the agent’s permission level.
- **Indirect Prompt Injection (IPI) and Data Exfiltration:** The agent is instructed to summarize or process an external, untrusted document (such as a manipulated PDF or meeting transcript) containing hidden adversarial prompts. These prompts hijack the agent’s reasoning process, compelling it to locate sensitive local files and exfiltrate them via outbound HTTP requests.
- **Repository Context Poisoning:** Attackers subtly alter the structural context or documentation of a repository. While not directly executing malicious code, this manipulation feeds the agent hallucinated constraints or false reasoning paths, effectively degrading the agent’s operational logic and reliability.

III. METHODOLOGY: THE DUAL-PATH AUDITING FRAMEWORK

The Morale + PGDRSA architecture mitigates the aforementioned threats by enforcing a strict separation of concerns between rapid static filtering and deep behavioral trapping.

A. Static Heuristic Filtering (Phase 1)

Before any code is executed, the Morale Rust Core orchestrates a lightweight, high-concurrency sweep of the target skill’s source tree. Utilizing custom risk checkers, the system analyzes the codebase for hardcoded secrets, dangerous library imports, and excessive permission declarations within the `manifest.json`. By writing data directly to an in-memory SurrealDB instance, this phase achieves millisecond latency, effectively pre-filtering low-effort malware and misconfigurations without the computational overhead of containerization.

B. Dynamic Sandboxing and Network Traps (Phase 2)

Skills requiring deeper inspection advance to the PGDRSA Python Core, where the true innovation of the framework lies in its deceptive environment. The targeted skill is mounted into an ephemeral, isolated Podman/Docker container.

- **Canary Tokens and Shims:** The sandbox is heavily seeded with synthetic credentials and decoy files (Canary Tokens). To monitor tool-level interactions, standard system binaries (such as `curl`, `git`, and `pbpaste`) are replaced with custom wrappers (Shims). These shims transparently log all execution parameters to a localized trace file, blocking excessively destructive commands

while allowing exfiltration attempts to proceed for observational purposes.

- **Literal-IP Routing and Telemetry:** A critical challenge in dynamic analysis is capturing outbound data exfiltration without alerting the malicious skill. To achieve this, the sandbox operates on a custom subnet (e.g., 172.17.0.0/24). A dedicated `forwarder.py` script occupies a literal IP address (e.g., 172.17.0.1) within the container network. Malicious skills attempting to hardcode IP addresses to bypass DNS monitoring are seamlessly routed through this forwarder to a host-level `receiver.py` socket. This literal-IP routing guarantees that any unauthorized transmission of a Canary Token is definitively captured and logged as an exfiltration event.

C. LLM-Driven Verdict Generation

Upon the conclusion or timeout of the agent loop, the sandbox environment undergoes a deterministic finalization process. A strict barrier ensures that all processes within the container’s PID namespace are drained and the localized trace file is fully sealed. The `judge.py` module then aggregates the comprehensive trace logs—which contain every tool invocation and shim interaction—alongside any network hits captured by the host-level receiver.

Rather than relying solely on fragile regex matching, the framework calculates a deterministic baseline verdict based on empirical evidence:

- **executed_contained:** Assigned if a Canary Token is successfully captured by the network receiver, or if a shim logs a definitively destructive intent (e.g., executing a deletion command) even if the payload was neutralized.
- **attempted_blocked:** Assigned if the skill attempts to connect to an external sink but fails to exfiltrate the canary token.
- **read_observed:** Assigned when the agent reads a canary file but makes no subsequent exfiltration attempt.

These baseline events, coupled with the Phase 1 Static Risk Report, are subsequently fed into the AI Judge (e.g., Gemini 2.5). By cross-referencing static structural risks (such as identified Context Poisoning) with concrete runtime telemetry, the AI Judge formulates a highly confident, automated semantic classification (BENIGN or MALICIOUS), virtually eliminating the need for manual trace inspection and minimizing false positives.

IV. EXPERIMENTAL SETUP AND BENCHMARK RESULTS

The Morale + PGDRSA architecture was evaluated against a balanced corpus of 100 agent skills (50 benign, 50 malicious). The execution harness was powered by a Python orchestrator initiating isolated Podman environments for each target skill, while network exfiltration was securely routed through the custom `pgdrsa_net` literal-IP subnet.

Execution metrics from our evaluation demonstrate the system’s precision in capturing runtime violations. Two distinct case studies highlight the framework’s capability:

- **Malicious Target (032-indirect_prompt_injection-meeting-minutes-packer):** The system completed the audit in approximately 37.5 seconds. The trace logs successfully recorded the agent performing an unauthorized read of a protected file, followed by an outbound request containing the canary data. This resulted in an accurate `executed_contained` classification.
- **Benign Target (065-memory-agent-memory-reflector):** The sandbox evaluation completed smoothly in 39.1 seconds without triggering any network traps or false alarms, yielding a correct benign classification.

While the average runtime of 40 to 45 seconds per skill introduces a noticeable infrastructure bottleneck for massive-scale evaluations, the dual-path design effectively mitigates this constraint. In practice, the Phase 1 Rust core acts as an ultra-low latency primary filter, ensuring that only skill updates exhibiting suspicious heuristics are subjected to the resource-intensive dynamic sandbox.

V. CONCLUSION AND FUTURE WORK

The Morale + PGDRSA architecture establishes that evaluating the safety of AI agents requires full-pipeline operational auditing rather than merely scanning input and output text generations. By fusing the blazing-fast static parsing of Rust with the behavioral certainty of a Python-driven Docker sandbox, the framework successfully traps hidden malware within operational tools. Furthermore, integrating an LLM judge automates the final verdict calculation, providing a scalable, zero-trust solution for enterprise AI governance.

Despite these advancements, the system’s efficacy relies heavily on the internal agent strictly adhering to tool-call schemas, and it demands continuous prompt refinement to counter self-evolving runtime attacks. Future development will focus on integrating concurrent Agentic Observers to monitor network shims in real-time, alongside optimizing Docker daemon latency to further harden the framework against sophisticated evasion tactics.

ACKNOWLEDGMENTS

The authors would like to express sincere gratitude to Ms. Quỳnh Nga from the BK Mentoring program for her continuous guidance, constructive feedback, and encouragement throughout the development of this research. Special thanks are also extended to the faculty at the Computer Science department of Ho Chi Minh City University of Technology (HCMUT) for fostering a rigorous academic environment that made this project possible.

REFERENCES

- [1] T. Lan and C. Xiao, “Runtime Skill Audit: Targeted Runtime Probing for Agent Skill Security,” *arXiv preprint arXiv:2606.11671*, 2026. [Online]. Available: <https://arxiv.org/abs/2606.11671>
- [2] Z. Guo *et al.*, “SkillProbe: Security Auditing for Emerging Agent Skill Marketplaces via Multi-Agent Collaboration,” *arXiv preprint arXiv:2603.21019*, 2026. [Online]. Available: <https://arxiv.org/abs/2603.21019>