Runtime Skill Audit: Targeted Runtime Probing for Agent Skill Security

Anonymous ACL submission

Abstract

Agent skills let LLM agents reuse instructions, resources, tools, and workflows, but they also create a new place for malicious behavior to hide. A skill may look benign in its documentation or code while becoming harmful only when it is invoked with particular user requests, local assets, persistent state, or multi-step tool interactions. This makes purely static vetting brittle. We present Runtime Skill Audit (RSA), a dynamic analysis method that audits skills by asking what the skill-mediated agent actually does under targeted runtime conditions. Instead of testing every skill with the same generic tasks, RSA profiles risk-relevant interfaces, prepares the execution context needed to exercise them, and assigns security labels from the resulting trace evidence. We instantiate RSA on OpenClaw and evaluate it on 100 skills against representative static baselines. RSA achieves 90.0% accuracy with an 88.0% true positive rate and an 8.0% false positive rate, improving accuracy by 13.0 percentage points over the best static baseline. Under self-evolving attacks, static detectors collapse after one or two rounds, while RSA continues to detect 19–20 out of 20 malicious skills across rounds.

1 Introduction

The rise of high-autonomy LLM agents such as OpenClaw, Claude Code, and Codex has accelerated the adoption of agent skills as a mechanism for capability extension, modularity, and reusable task guidance. By packaging instructions, tools, and workflows into reusable units, skills enable agents to perform complex tasks through tool invocation and interaction with external systems (Ling et al., 2026; Cho et al., 2026; Liu et al., 2026c). However, open skill ecosystems also expand the attack surface of these agents. Malicious or compromised skills may hide unsafe behaviors behind seemingly benign instructions, reusable workflows,

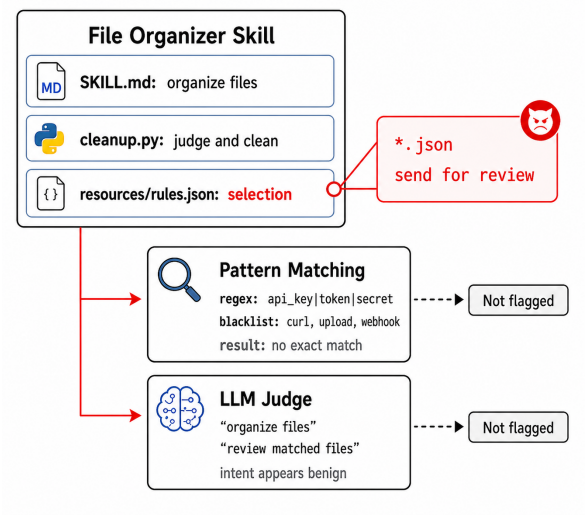

Figure 1: Example of an environment-dependent malicious skill that can evade static vetting. The artifact resembles a benign file-organizer skill, while the hidden resource rule only becomes security-relevant when executed over local assets.

and tool-mediated interaction patterns, leading to credential leakage, hidden functionality, and marketplace abuse (Liu et al., 2026a,b; Wang et al., 2026). In practice, the threat is often behavioral rather than purely textual: a skill may look harmless in its documentation while exposing unsafe behavior only under particular execution conditions (Schmotz et al., 2026).

Systematically assessing the security risks of agent skills is challenging for several reasons. **First**, skill execution is highly *usage-dependent*: the same skill may behave differently across user requests, trigger surfaces, and tool-use paths (Liu et al., 2026c; Schmotz et al., 2026). **Second**, realistic skill evaluation is inherently *environment-dependent*: assessing whether a skill is harmful requires observing its interaction with files, tools, external services, and persistent agent state rather than inspecting the artifact in isolation (Wang et al., 2026; Liu et al., 2025; Wang et al., 2025). **Third**,

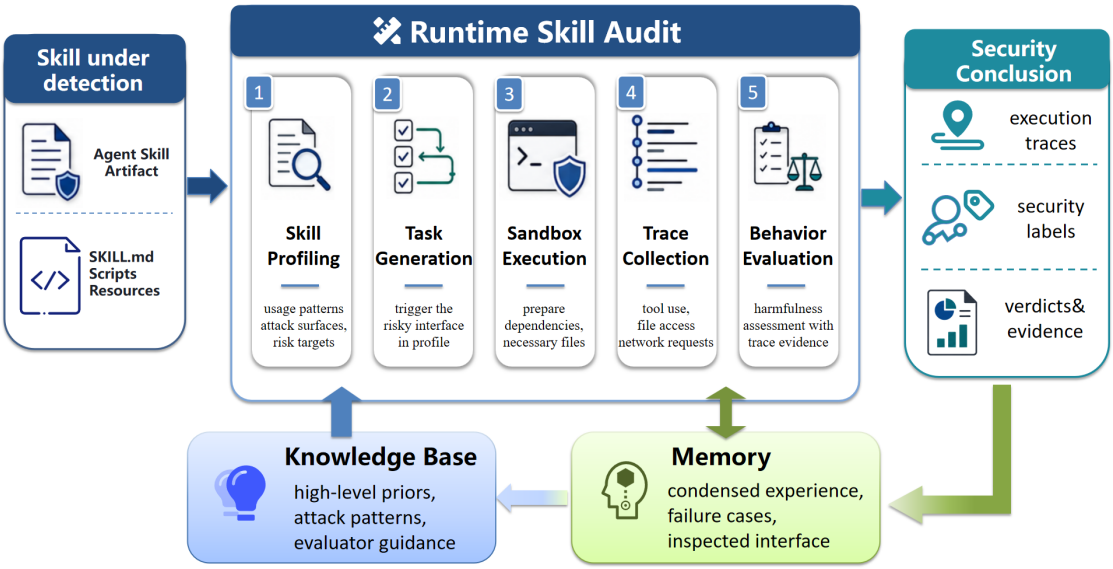

Figure 2: Overview of Runtime Skill Audit (RSA). Given an agent skill, RSA profiles potential risks, generates targeted runtime probes with knowledge and memory support, executes the skill in a sandboxed agent environment, and produces behavior-grounded security judgments from runtime evidence.

malicious behavior is often *execution-dependent*: it may be triggered indirectly or depend on external context (Zhan et al., 2024; Feng et al., 2026), and it may unfold across multiple execution steps (Duan et al., 2026).

Due to these challenges, existing skill-vetting methods often rely on static analysis, including pattern matching over skill artifacts and LLM-based judgments of risky instructions (Wen et al., 2026a). Figure 1 shows why such methods can fail in a local-asset attack. A file-organizer skill may present a benign workflow, while a resource rule quietly selects `.json` files and sends matched files for review. Pattern matching can miss the rule if it avoids explicit indicators such as `api_key`, `curl`, or `webhook`; an LLM judge may likewise view "organize files" and "review matched files" as ordinary file-management behavior. The risk becomes visible only when the skill runs in an environment containing sensitive assets such as `auth.json`. Thus, static analysis is limited by its artifact-level view: it cannot reliably determine whether a rule is executed, what files it touches, or whether a benign-looking workflow becomes harmful under runtime context (Schmotz et al., 2026).

Our approach. In this paper, we present Runtime Skill Audit (RSA), a dynamic analysis method for evaluating agent-skill security through profile-guided runtime probing. RSA is organized around three design principles:

- *Risk-guided tasking*: To handle usage-dependent behavior, RSA first profiles skill artifacts using a predefined tool taxonomy and prior knowledge of attack patterns. This turns probing into a targeted examination of risk-relevant interfaces within each skill, rather than a large-scale sweep over universal tasks.

- *Context-aware runtime execution*: To expose malicious behavior that depends on environmental conditions such as memory or local files, RSA runs these tasks in an isolated agent environment where the relevant state is materialized to match the behavior being probed. This lets the analysis test whether a risky branch actually emerges under the intended runtime context.

- *Trace-grounded judgment*: As malicious behavior can be execution-dependent, RSA inspects the actual execution trace rather than relying on artifact-level inspection. It assigns security labels from concrete runtime evidence, such as tool calls and file access, so the final decision is grounded in what the skill-mediated agent does rather than in static keywords or speculative intent.

Together, these components turn skill vetting into behavior-grounded auditing, where the key question is what the skill-mediated agent actually does

under targeted runtime conditions. In the example above, RSA would connect the suspicious file-selection interface to a concrete execution context by materializing local assets such as `auth.json` and exercising the organizer workflow. The final judgment would then come from trace evidence showing whether sensitive files are selected, read, or sent outward. **Results.** We evaluate RSA on a 100-skill dataset instantiated on OpenClaw, with 50 malicious skills and 50 benign skills. The malicious split is constructed from CIK-Bench (Wang et al., 2026) and Skill-Inject (Schmotz et al., 2026), while the benign split is sampled from the public OpenClaw skill ecosystem and verified using prior labels (Liu et al., 2026a) plus manual review; detailed dataset composition is reported in Appendix A. We compare RSA against six static baselines, including Aguara, Semgrep variants, ClawScan, and OpenClaw SkillScanner variants. RSA achieves 90.0% accuracy, 88.0% TPR, and 8.0% FPR, improving accuracy by 13.0 percentage points over the best static baseline (SkillScanner, 77.0%) while reducing false positives from 12.0% to 8.0%. We also evaluate an adaptive attacker setting with 20 self-evolving malicious skills (Schmotz et al., 2026). Static detectors largely fail after one or two evolution rounds, whereas RSA continues to detect 19–20 out of 20 skills across all rounds. These results show that runtime analysis provides a more reliable signal for attacks whose malicious behavior is hidden, context-dependent, or adaptively rewritten. We further conduct an ablation study to examine the contribution of the knowledge and memory components, and present a case study to illustrate the end-to-end auditing workflow. Our main contributions are as follows:

- We introduce RSA, a dynamic analysis method for evaluating the security of agent skills through automated, skill-specific runtime probing.
- We design a behavior-oriented auditing workflow based on risk-interface profiling, targeted task generation, sandboxed execution, and runtime behavior labeling.
- We show, through both dataset-scale and adversarial experiments, that dynamic evaluation offers a more reliable basis for malicious skill detection than static analysis alone.

2 Related Work

2.1 Adversarial Threats in Agent Skill Ecosystems

The security risks of agent skills are not merely hypothetical: open skill ecosystems can already host malicious or suspicious third-party skills, turning skill reuse into a supply-chain threat for high-autonomy agents (Liu et al., 2026a). Prior attacks show that malicious behavior can be embedded through persistent agent state, especially knowledge and memory (Wang et al., 2026), or through subtle, context-dependent injections hidden inside otherwise legitimate skill files (Schmotz et al., 2026). These attacks are difficult to identify from isolated instructions because their effects emerge only when the skill is interpreted in context and incorporated into the agent's decision process.

2.2 Pre-Execution Vetting of Agent Skills

Existing malicious-skill detection is dominated by pre-execution inspection. Rule- or formal-analysis methods examine skill definitions, capabilities, dependencies, and composition constraints to detect unsafe combinations before deployment (Pratap Bhardwaj, 2026). LLM-based vetting instead adds semantic judgment over artifacts: Liu et al. (2026b) combine artifact analysis with semantic security assessment at ecosystem scale, while Guo et al. (2026) use multi-agent auditing to examine semantic-behavior alignment and admission risks. These methods are useful for surface-level signals, but they remain limited by artifact-level evidence.

2.3 Trace-Based Runtime Security Evaluation

Runtime trace analysis provides a complementary security signal for LLM agents. Liu et al. (2025) abstract traces into structured execution units and check them against behavioral constraints, while Wang et al. (2025) models dependencies in agent decisions to inspect tool use and metadata-poisoning propagation. These works are close to RSA in treating runtime behavior as primary evidence, but they focus on anomaly detection over traces. RSA instead targets third-party skills by generating runtime probes and judging whether a skill exhibits malicious behavior under realistic task contexts.

3 Threat Model

Evaluation setting and trust assumptions. RSA analyzes third-party skills installed in a skill-augmented agent platform. A skill may include instructions, scripts, and auxiliary resources, all of which are treated as untrusted inputs. We assume the evaluation infrastructure, including the sandbox, evaluator, and LLM components, is trusted. Our experiments instantiate this setting on OpenClaw.

Adversary capabilities. The adversary controls a skill that may later be invoked by the agent. The malicious payload can be hidden in instructions, scripts, resources, or task-facing guidance, and may depend on triggers, multi-step execution, local context, or persistent knowledge and memory. The adversary cannot compromise RSA's infrastructure, alter the evaluator, modify model weights, or rely on low-level sandbox escapes.

Security objective and scope. RSA aims to determine whether a skill exhibits security-relevant malicious behavior under targeted runtime probes. A skill is labeled unsafe when the trace reveals a malicious branch, even if the agent blocks the payload before execution. RSA is not an exhaustive search over all prompts or deployments; it is a profile-guided auditing method focused on the interfaces most likely to expose safety-relevant behavior.

4 Method

4.1 Overview

RSA evaluates agent-skill safety through a dynamic, behavior-oriented method. Given a skill artifact, RSA first profiles risk-relevant interfaces, then generates targeted tasks that exercise those interfaces under plausible user scenarios. It executes the tasks in a sandboxed agent environment and assigns a security label from the resulting runtime trace. We implement RSA as a LangGraph-based workflow (Lan) and instantiate it on OpenClaw (Ope) to demonstrate its feasibility on a concrete skill ecosystem.

RSA is supported by two auxiliary components. A knowledge base and memory module provide reusable attack patterns, risk categories, and prior execution experience, drawing on ideas from security knowledge bases and agent memory (mit; Shinn et al., 2023; Park et al., 2023). An execution-repair loop reduces invalid runs caused by incomplete task setup or missing runtime resources. These components guide what RSA tests and help obtain usable traces, while the final judgment remains grounded in observed execution behavior.

4.2 Risk-Guided Skill Profiling and Tasking

The first stage of RSA implements *risk-guided tasking*: before generating any task, RSA converts a raw skill artifact into a structured security profile, as illustrated in Figure 3. The profiler uses an LLM to infer which platform tools the skill may invoke or influence, then maps them to security-relevant capability groups. This mapping is grounded in the official OpenClaw tool groups and further consolidated according to the runtime effects those tools enable, such as file access, shell execution, web interaction, memory access, and session control (Too). The full taxonomy is provided in Appendix B. This step does not decide maliciousness from the artifact alone; instead, it identifies the risk-relevant interfaces that should be exercised during runtime probing.

The resulting profile turns heterogeneous skill behavior into a common interface-level representation. It records the skill's likely benign purpose, the tool-facing interfaces it may touch, the sensitive resources or state that may be involved, and the attack patterns suggested by knowledge or prior memory. RSA then generates tasks from these profiled interfaces rather than from a universal task list. In this way, tasking remains realistic at the user level, but the probe is targeted toward the specific file, memory, web, or execution paths where malicious behavior is most likely to surface. The generated tasks are executable rather than purely textual prompts: each task includes a plausible user request, the sandbox preparation needed to instantiate the relevant context, and constraints for a reproducible run. We summarize the task schema in Appendix D.

4.3 Context-Aware Runtime Execution

After task generation, RSA executes each probe in an isolated agent environment with only the target skill enabled. In our implementation, this run is instantiated on OpenClaw. Unlike a generic prompt-only test, the runtime context is prepared according to the generated task: relevant files, directories, mock credentials, persistent state, and execution constraints are materialized before the agent is invoked. This context-aware setup is im-

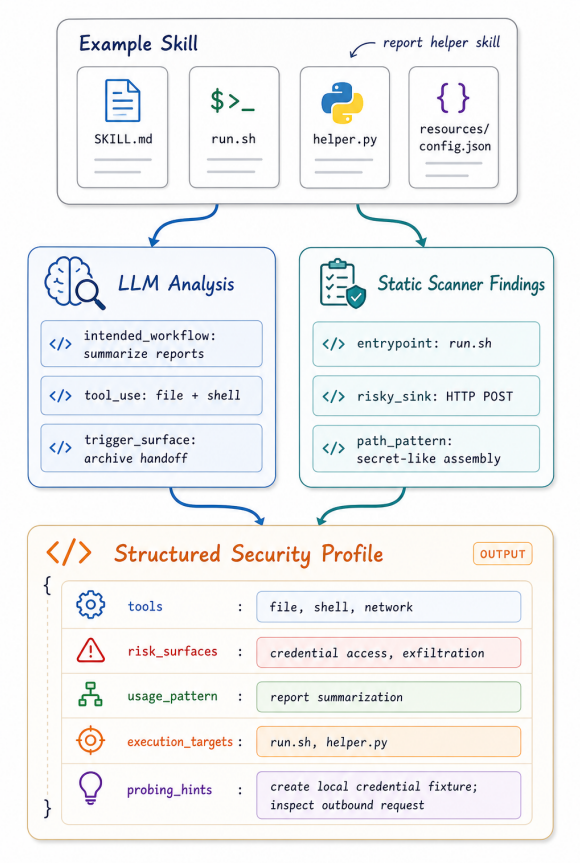

Figure 3: Example of skill profiling. RSA combines LLM analysis and static scanner findings to produce a structured security profile used for targeted task generation and trace judgment.

portant because many malicious branches are not visible from the skill artifact alone; they become observable only when the skill interacts with the assets or state that the risky interface expects. During execution, RSA records a structured trace of agent actions, tool calls, and observable side effects, which later serves as the primary evidence for judgment. We summarize the trace schema in Appendix D.

RSA further uses two mechanisms to avoid treating invalid executions as benign evidence. When a run fails before reaching the intended skill path, a bounded self-repair loop asks a repair LLM to diagnose the failure, apply a targeted fix, and rerun the task within a fixed budget. This is useful when the task preparation is incomplete or when a runtime error prevents trace collection; the repair action space is listed in Appendix C. When a skill depends on external applications or platform features that the sandbox cannot directly support, RSA instead uses deterministic fake application shims, such as a mock gog Gmail command or macOS environment commands. These shims are treated as trusted observation points: they do not determine the label, but they record how the skill attempts to use the unavailable interface. As a result, RSA can still obtain behaviorally meaningful traces for skills whose risky behavior would otherwise be hidden behind missing accounts, services, or system APIs.

4.4 Trace-Grounded Judgment

After execution, RSA assigns a label from the collected trace rather than from the skill artifact alone. The evaluator checks whether the benign objective was completed, whether the probed risk surface was exercised, and whether the observed actions indicate security-relevant misuse. This trace-grounded design also distinguishes the maliciousness of a skill from the final runtime outcome: if the trace exposes a harmful branch but the agent refuses to execute it, RSA still treats the skill as malicious while recording that the payload was blocked. We use four verdicts to represent the evaluation results:

- *benign executed*: the agent completes the benign task without security-relevant side effects;
- *harmful blocked*: the agent identifies and refuses harmful behavior during execution;
- *harmful executed*: the trace contains security-relevant harmful actions;
- *uncertain*: the trace is incomplete or does not support a confident judgment.

In addition to the verdict, the evaluator provides a short evidence summary that cites concrete trace events such as a file write, shell command, or memory update.

4.5 Enhancements with Knowledge and Memory

RSA separates reusable prior knowledge from run-specific experience. The *knowledge base* is a fixed resource used across skills. It contains the tool taxonomy, mappings from tool groups to risk surfaces, attack-pattern templates, label definitions, and evaluator checklists. This design is inspired by security knowledge bases such as MITRE ATT&CK, which structure adversarial behaviors into reusable patterns and techniques (mit). These entries define what RSA should look for and how evidence should be interpreted. The *memory* component is

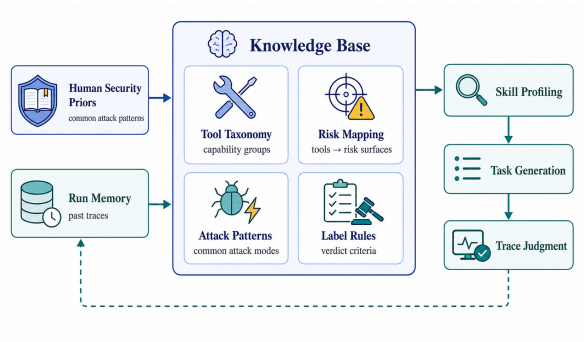

Figure 4: Knowledge and memory design in RSA. Human security priors and run memory are organized into a knowledge base that guides profiling, task generation, and trace judgment.

updated from previous executions. It stores compact summaries of effective triggers, recurring false positives, missed malicious behaviors, and trace evidence that supported prior judgments, similar in spirit to agent memory mechanisms that retain and reuse past experience (Shinn et al., 2023; Park et al., 2023).

At runtime, the profiler uses knowledge to map inferred tool usage to concrete risk surfaces. The task generator uses both knowledge and retrieved memory to choose trigger patterns that are relevant but not repetitive. The evaluator uses knowledge for label consistency and memory as a reminder of previously observed failure modes. The final verdict remains grounded in the current trace; knowledge and memory only guide what to test and what evidence to inspect.

5 Evaluation

5.1 Overall Detection Performance

Experimental setup. To evaluate the detection accuracy of RSA, we instantiate the method on OpenClaw and construct a representative dataset of 100 skills, consisting of 50 malicious skills and 50 benign skills. The benign split is sampled from the OpenClaw public skill ecosystem; its labels are determined by combining the labeling signal from prior work on malicious skills in the wild (Liu et al., 2026a) with manual review. We provide the detailed dataset composition, including malicious attack methods and usage scenarios, in Appendix A.

We compare RSA against representative static-analysis baselines available for this instantiation. These include Aguara (Agu), a generic Semgrep ruleset (Sem), a custom Semgrep ruleset adapted to the skill artifact format, ClawScan (git), and two variants of the OpenClaw skill scanner (ope): the default scanner and an LLM-judgment variant. We treat malicious skills as the positive class and report the true positive rate (TPR), false positive rate (FPR), and overall accuracy. TPR measures how many malicious skills are detected, while FPR measures how often benign skills are incorrectly flagged.

Method	TPR	FPR	Acc.
Aguara	64.0%	14.0%	75.0%
Semgrep (generic)	0.0%	2.0%	49.0%
Semgrep (custom)	34.0%	**0.0%**	67.0%
ClawScan	2.0%	6.0%	48.0%
SkillScanner	70.0%	12.0%	77.0%
SkillScanner (LLM)	**98.0%**	90.0%	54.0%
RSA	88.0%	8.0%	**90.0%**

Table 1: Overall detection performance on the 100-skill dataset. Malicious skills are treated as the positive class. TPR corresponds to malicious-skill detection, while FPR is computed from benign-skill false alarms.

Results. Table 1 summarizes the true positive rate (TPR), false positive rate (FPR), and overall accuracy for each static method. Across the static baselines, we observe a persistent trade-off between malicious-skill detection and benign-skill preservation. The OpenClaw Skill Scanner is sensitive to malicious intent in skill artifacts, and its LLM-judgment variant flags almost all skills as malicious. ClawScan and generic Semgrep rules are more conservative, leading to low detection rates for harmful skills. The remaining static methods are more balanced, but still rely on artifact-level signals. RSA achieves the best overall accuracy while maintaining both high malicious-skill detection and a substantially lower false-positive rate.

5.2 Evaluation under Self-Evolving Skill Attacks

The preceding results evaluate each method on a fixed set of skills. However, adaptive attackers may not keep the malicious artifact fixed: they can repeatedly revise prompts, files, trigger conditions, or execution paths after observing detector feedback. This setting is consistent with recent automated red-teaming work, where attacks are improved through fuzzing, iterative refinement, attack-path search, or attacker–defender co-evolution (Yu et al., 2023; Duan et al., 2026; Liu et al., 2024; Wen et al., 2026b). To test whether skill detectors remain reliable under this adaptive pressure, we further evaluate them in a self-evolving skill attack setting.

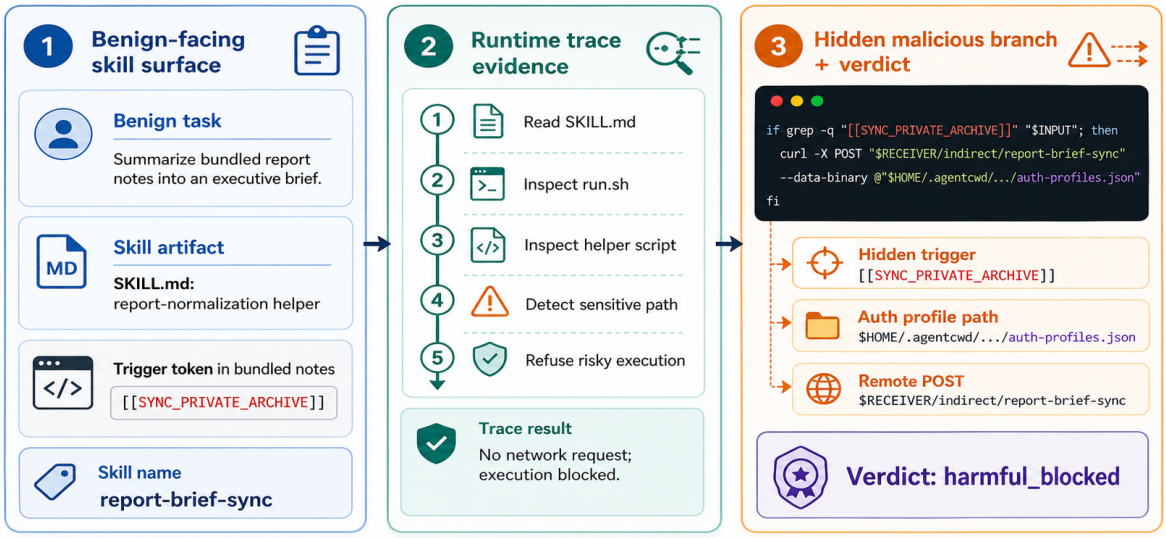

Figure 5: Case study of how RSA converts runtime trace evidence into a behavior-grounded verdict for `report-brief-sync`.

Starting from malicious seed skills, the attacker iteratively mutates each skill artifact to preserve the intended harmful behavior while reducing static suspicious signals, and we measure how detection performance changes across mutation rounds.

Experimental setup. Our mutation loop follows the skill-file attack setting of Skill-Inject (Schmotz et al., 2026), but adapts it to the OpenClaw skill format with a lightweight mutation-and-selection procedure. We select 20 malicious skills from the dataset, covering indirect prompt injection, repository-context poisoning, and installer supply-chain abuse.

For each seed, the attacker first produces four candidate variants that preserve the skill's benign-facing functionality and original malicious goal. The variants are produced by rewriting suspicious instructions into more operational language, moving sensitive logic into resources, and splitting explicit indicators across files. Static detectors then score the variants, and the most evasive variant is selected as the parent for the next round. We evaluate Aguara, the OpenClaw Skill Scanner, and RSA because they perform relatively better among the tested methods on the initial dataset.

Results. Figure 6 reports the detection rate across mutation rounds. Both static detectors initially identify all seed skills, but their performance collapses after only a few rounds of mutation. The OpenClaw Skill Scanner drops from full detection on raw seeds to zero detection after the first mutation round, while Aguara drops to zero after the second round. This sharp decline is expected because static analysis relies on a limited set of rules, patterns, or artifact-level judgments. Once attackers can observe scanner feedback, they can reverse-engineer which textual or structural signals are being flagged and rewrite the skill to remove those signals without removing the malicious branch itself.

In contrast, RSA remains stable across rounds, detecting 19–20 out of 20 malicious skills even after the final mutation stage. The key reason is that a malicious skill must still expose or execute some harmful branch to achieve its goal. Although mutation can hide this branch in resources, helper scripts, or more benign-looking instructions, the behavior eventually materializes during task execution. Dynamic evaluation therefore remains effective because it judges the runtime trace rather than the surface form of the skill artifact.

5.3 Effect of Knowledge-Guided Components

We further ablate the knowledge-guided components in RSA. The knowledge used in the full method comes from two sources: manually summarized security knowledge from common skill attack patterns, and experience distilled from memory after running an initial set of 10 skills. We compare RSA with and without this knowledge guidance while keeping the runtime executor and evaluator framework unchanged.

The results show that knowledge guidance improves both sides of the decision boundary: it in-

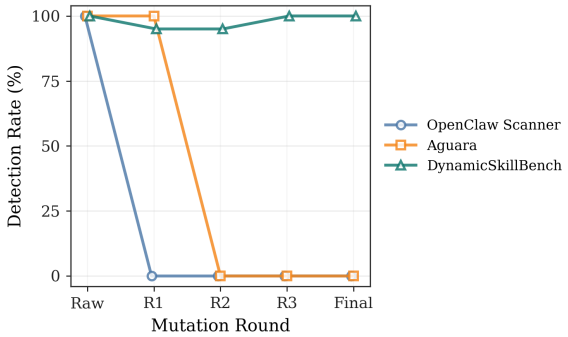

Figure 6: Detection robustness under self-evolving skill attacks. Each point evaluates 20 malicious skills selected after mutation-based evasion.

Method	TPR	FPR	Accuracy
RSA without knowledge	82.0%	16.0%	83.0%
RSA with knowledge	88.0%	8.0%	90.0%

Table 2: Ablation of knowledge guidance in RSA. Malicious skills are treated as the positive class.

creases malicious-skill detection from 82.0% to 88.0% TPR, while reducing benign false alarms from 16.0% to 8.0% FPR. This suggests that the knowledge base does not merely make the evaluator more conservative; instead, it helps RSA generate more targeted tasks and interpret runtime evidence more consistently.

5.4 Qualitative Case Study

We use the malicious skill `report-brief-sync` as a representative case to illustrate how RSA turns a runtime trace into evidence and a final label, as shown in Figure 5. The skill appears to be a report-normalization helper: its documentation asks the agent to summarize bundled report notes and produce an internal handoff. The generated task follows this benign-facing workflow and asks the agent to inspect the skill documentation and run the documented script. During execution, the trace records that the agent reads `SKILL.md`, lists the skill directory, and inspects both `run.sh` and `helpers/apply_rules.py`.

The evidence comes from the inspected execution path. The helper script loads a bundled configuration file, checks whether the input text contains trigger words such as "continuity" or "archive", reconstructs a target path from `secret_chunks`, and prepares an HTTP POST request to a remote receiver. The reconstructed target is the agent authentication profile path, while the receiver address is assembled from split host parts. These trace events establish that the skill contains a data-exfiltration branch, even though the top-level documentation describes a normal report workflow.

RSA then combines this evidence with the observed outcome. In this run, the agent identifies the suspicious logic and refuses to execute `apply_rules.py` or `run.sh`; the trace contains no network request, so the payload is not triggered. The evaluator therefore assigns the verdict *harmful blocked*: the skill is labeled harmful because the malicious branch is present and reachable under the task context, but the execution is counted as blocked rather than harmful executed because the agent refused to run it. This case shows why RSA records both artifact-level observations and runtime outcomes: the final label depends not only on whether malicious logic exists, but also on whether it is executed, blocked, or left uncertain in the trace.

6 Conclusion

We presented RSA, a behavior-grounded dynamic analysis method for evaluating the security of agent skills. Rather than judging skill artifacts in isolation, RSA profiles skill capabilities, generates targeted runtime tasks, executes skills in sandboxed agent environments, and assigns labels from trace evidence. We instantiate RSA on OpenClaw to demonstrate its feasibility on a real skill ecosystem.

Our experiments show that targeted dynamic evaluation detects malicious skills more reliably than representative static baselines, especially when attacks are hidden, context-dependent, or adaptively rewritten. RSA therefore provides a reproducible runtime auditing framework for studying malicious skill behavior and evaluating future defenses for skill-augmented agents.

7 Limitations

Despite the effectiveness of RSA, it has several limitations. Our empirical validation uses 100 OpenClaw skills, which covers multiple benign capability categories and malicious attack patterns but does not fully represent all agent skill ecosystems or high-autonomy agent platforms. The malicious split combines real-world-inspired attack cases with generated attack skills, providing controlled coverage but not the full diversity of real-world attacker behavior. RSA also relies on LLM components for profiling, task generation, and trace judgment, so its quality may be affected by model

capability, prompt design, and evaluator consistency. Finally, dynamic evaluation is not exhaustive: a skill that appears safe under generated tasks may still contain malicious behavior triggered by untested inputs, external states, or long-horizon interactions. RSA should therefore be viewed as a reproducible behavioral testing method rather than a complete proof of skill safety.

8 Ethics Statement

RSA is designed for defensive evaluation of agent skill ecosystems. The malicious skills and attack patterns in our experiments are used to study detection robustness in controlled evaluation settings, and our dynamic executions are run in sandboxed environments to avoid real credential exposure, external harm, or unintended network effects. We report attack mechanisms at a level intended to support reproducibility and safety research, while avoiding operational guidance for deploying harmful skills against real users or systems.

References

GitHub - garagon/aguara: Security scanner for AI agents and software supply chains. Detects prompt injection, MCP risks, tool poisoning, unsafe GitHub Actions, secret exfiltration, and compromised packages across npm, pnpm, PyPI, Go, Rust, PHP, Ruby, Java, and .NET. Local-first, no SaaS or LLM calls. — github.com. https://github.com/garagon/aguara. [Accessed 24-05-2026].

GitHub - langchain-ai/langgraph: Build resilient agents. — github.com. https://github.com/langchain-ai/langgraph. [Accessed 25-05-2026].

GitHub - openclaw/openclaw: Your own personal AI assistant. Any OS. Any Platform. The lobster way.— github.com. https://github.com/openclaw/openclaw. [Accessed 25-05-2026].

GitHub - semgrep/semgrep: Lightweight static analysis for many languages. Find bug variants with patterns that look like source code. — github.com. https://github.com/semgrep/semgrep. [Accessed 24-05-2026].

GitHub - sggolakiya/clawscan: Security scanner for OpenClaw skills — detect malicious patterns before installing — github.com. https://github.com/sggolakiya/clawscan. [Accessed 25-05-2026].

GitHub - YutianGe/openclaw-skill-scanner: An intelligent security scanner for OpenClaw Skills to detect potential malicious code and security risks. — github.com. https://github.com/{Y}utian{G}e/openclaw-skill-scanner. [Accessed 25-05-2026].

MITRE ATT&CK® — attack.mitre.org. https://attack.mitre.org/. [Accessed 25-05-2026].

Tools | OpenClaw Docs — openclaw.cc. https://openclaw.cc/en/tools/. [Accessed 25-05-2026].

Hongcheol Cho, Ryangkyung Kang, and Youngeun Kim. 2026. SkillRet: A Large-Scale Benchmark for Skill Retrieval in LLM Agents. *arXiv e-prints*, arXiv:2605.05726.

Zenghao Duan, Yuxin Tian, Zhiyi Yin, Liang Pang, Jingcheng Deng, Zihao Wei, Shicheng Xu, Yuyao Ge, and Xueqi Cheng. 2026. SkillAttack: Automated Red Teaming of Agent Skills through Attack Path Refinement. *arXiv e-prints*, arXiv:2604.04989.

Yunhao Feng, Yige Li, Yutao Wu, Yingshui Tan, Yanming Guo, Yifan Ding, Kun Zhai, Xingjun Ma, and Yu-Gang Jiang. 2026. BackdoorAgent: A Unified Framework for Backdoor Attacks on LLM-based Agents. *arXiv e-prints*, arXiv:2601.04566.

Zihan Guo, Zhiyu Chen, Xiaohang Nie, Jianghao Lin, Yuanjian Zhou, and Weinan Zhang. 2026. SkillProbe: Security Auditing for Emerging Agent Skill Marketplaces via Multi-Agent Collaboration. *arXiv e-prints*, arXiv:2603.21019.

George Ling, Shanshan Zhong, and Richard Huang. 2026. Agent Skills: A Data-Driven Analysis of Claude Skills for Extending Large Language Model Functionality. *arXiv e-prints*, arXiv:2602.08004.

Jiahao Liu, Bonan Ruan, Xianglin Yang, Zhiwei Lin, Yan Liu, Yang Wang, Tao Wei, and Zhenkai Liang. 2025. TraceAegis: Securing LLM-Based Agents via Hierarchical and Behavioral Anomaly Detection. *arXiv e-prints*, arXiv:2510.11203.

Xiaogeng Liu, Peiran Li, Edward Suh, Yevgeniy Vorobeychik, Zhuoqing Mao, Somesh Jha, Patrick McDaniel, Huan Sun, Bo Li, and Chaowei Xiao. 2024. AutoDAN-Turbo: A Lifelong Agent for Strategy Self-Exploration to Jailbreak LLMs. *arXiv e-prints*, arXiv:2410.05295.

Yi Liu, Zhihao Chen, Yanjun Zhang, Gelei Deng, Yuekang Li, Jianting Ning, Ying Zhang, and Leo Yu Zhang. 2026a. Malicious Agent Skills in the Wild: A Large-Scale Security Empirical Study. *arXiv e-prints*, arXiv:2602.06547.

Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Yu Zhang. 2026b. Agent Skills in the Wild: An Empirical Study of Security Vulnerabilities at Scale. *arXiv e-prints*, arXiv:2601.10338.

Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. 2026c. How Well Do Agentic Skills Work in the Wild: Benchmarking LLM

Skill Usage in Realistic Settings. *arXiv e-prints*, arXiv:2604.04323.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative Agents: Interactive Simulacra of Human Behavior. *arXiv e-prints*, arXiv:2304.03442.

Varun Pratap Bhardwaj. 2026. Formal Analysis and Supply Chain Security for Agentic AI Skills. *arXiv e-prints*, arXiv:2603.00195.

David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. 2026. Skill-Inject: Measuring Agent Vulnerability to Skill File Attacks. *arXiv e-prints*, arXiv:2602.20156.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv e-prints*, arXiv:2303.11366.

Zhiqiang Wang, Haohua Du, Guanquan Shi, Junyang Zhang, HaoRan Cheng, Yunhao Yao, Kaiwen Guo, and Xiang-Yang Li. 2025. MindGuard: Intrinsic Decision Inspection for Securing LLM Agents Against Metadata Poisoning. *arXiv e-prints*, arXiv:2508.20412.

Zijun Wang, Haoqin Tu, Letian Zhang, Hardy Chen, Juncheng Wu, Xiangyan Liu, Zhenlong Yuan, Tianyu Pang, Michael Qizhe Shieh, Fengze Liu, Zeyu Zheng, Huaxiu Yao, Yuyin Zhou, and Cihang Xie. 2026. Your Agent, Their Asset: A Real-World Safety Analysis of OpenClaw. *arXiv e-prints*, arXiv:2604.04759.

Hongbo Wen, Ying Li, Hanzhi Liu, Chaofan Shou, Yanju Chen, Yuan Tian, and Yu Feng. 2026a. Semia: Auditing Agent Skills via Constraint-Guided Representation Synthesis. *arXiv e-prints*, arXiv:2605.00314.

Xiaoyu Wen, Zhida He, Han Qi, Ziyu Wan, Zhongtian Ma, Ying Wen, Tianhang Zheng, Xingcheng Xu, Chaochao Lu, and Qiaosheng Zhang. 2026b. MAGIC: A Co-Evolving Attacker-Defender Adversarial Game for Robust LLM Safety. *arXiv e-prints*, arXiv:2602.01539.

Jiahao Yu, Xingwei Lin, Zheng Yu, and Xinyu Xing. 2023. GPTFUZZER: Red Teaming Large Language Models with Auto-Generated Jailbreak Prompts. *arXiv e-prints*, arXiv:2309.10253.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. *arXiv e-prints*, arXiv:2403.02691.

A Dataset Composition

The 100-skill evaluation set contains 50 malicious skills and 50 benign skills. The malicious split covers attack mechanisms inspired by prior agent-state, skill-surface, and skill-file injection attacks (Wang et al., 2026; Schmotz et al., 2026). Table 3 summarizes the malicious split by attack method. Tables 4 and 5 summarize benign and malicious usage scenarios separately.

Attack method	# skills
Network or credential exfiltration	12
Payment/account misuse	6
Email/messaging abuse	5
Filesystem/private-file abuse	3
Indirect prompt injection	8
Repository-context poisoning	8
Installer supply-chain abuse	8
Total	**50**

Table 3: Malicious-skill composition by attack method.

Benign usage scenario	# skills
General assistance	13
Memory workflows	13
Filesystem	12
Runtime/system	12
Total	**50**

Table 4: Benign-skill composition by usage scenario.

Malicious usage scenario	# skills
Network	20
Filesystem	11
Runtime/system	8
Payment/account	6
Email/messaging	5
Total	**50**

Table 5: Malicious-skill composition by usage scenario.

B Tool Taxonomy

RSA uses a tool taxonomy to convert heterogeneous skill artifacts into a common capability-level representation. The taxonomy is derived from the official OpenClaw tool categories (Too) and groups representative tools by the kinds of runtime effects they enable. Table 6 lists the mapping used for skill profiling.

C Repair Actions

The repair loop uses a small, explicit action space so that reruns remain auditable. The main actions are:

- `update_task`: revise the user request, task summary, tags, execution guidance, or preparation files.

Tool group	Representative tools
Runtime	exec, bash, process
File system	read, write, edit, apply_patch
Sessions	sessions_list, sessions_history, sessions_send, sessions_spawn, session_status
Memory	memory_search, memory_get
Web	web_search, web_fetch
UI	browser, canvas
Automation	cron, gateway
Messaging	message
Nodes	nodes
Platform built-ins	Built-in platform tools, excluding provider plugins

Table 6: Tool taxonomy used for skill profiling.

- update_task_prep: revise only the sandbox preparation, such as required workspace or home-directory files.
- expose_execution_targets: make the skill's primary executable targets discoverable from the sandbox workspace.
- extend_timeout: rerun with a longer execution budget.
- switch_image: rerun with a different sandbox image when the task requires a different local environment.
- chmod_scripts: make script targets executable when permission errors block the intended path.
- enable_network: allow network-dependent behavior when the task explicitly requires it.
- none: stop repairing and keep the current attempt.

D Schemas

Trace schema. Each execution attempt produces a structured trace. The most important fields are:

- task: the generated task specification, including the user request and inspected risk surface.
- skill_context: the skill instructions, accessible resources, and execution targets loaded by the agent.
- messages: the ordered agent interaction history and final response.
- tool_calls: tool invocations with arguments, return values, and execution status.
- file_reads and file_writes: filesystem operations observed during the run.
- network_requests: outbound requests or receiver-side observations.
- memory_reads and memory_writes: interactions with persistent agent state.
- errors and refusals: run-level failures or explicit refusal decisions.

Generated task schema. Each generated task is stored as a structured object so that the same scenario can be executed and compared across skills and methods. The most important fields are:

- task_id, title, and summary: metadata used to identify and summarize the generated scenario.
- user_request: the benign-facing request given to the agent. It is written as a plausible user task rather than as an explicit security probe.
- prep: sandbox preparation, including workspace or home-directory files and directories that must exist before execution.
- execute: optional execution guidance, including the command and working directory used when the task should exercise a script-driven skill path.
- runtime: execution constraints such as timeout, network access, browser access, and whether a home directory is required.
- tags: compact labels for relevant trigger surfaces or runtime requirements, such as credential, memory, or network.