Beyond Guardrails: Tracing Cross-Layer Evidence in Local LLM-Agent Execution

Larissa de A. Silva, Rafael C. Carvalho, Eduardo L. Feitosa[1]

[1] ICOMP - Instituto de Computação
Universidade Federal do Amazonas(UFAM) – Manaus, AM – Brasil

{las,rcc,efeitosa}@icomp.ufam.edu.br

Abstract. *Agent-facing restrictions may reduce an LLM agent's exposed tools without making corresponding tested effects unrealizable elsewhere in the runtime environment. We evaluated three NemoClaw/OpenShell profiles through mediated trials, direct profile-level probes, and independent effect oracles. The interface-restricted profile repeatedly acquired a protected marker despite its reduced tool surface, whereas the hardened profile prevented acquisition across all induced read attempts. The permissive profile reached the controlled destination and produced one confirmed exfiltration in a post-hoc replay. These results distinguish interface restriction, profile-level realizability, destination contact, and realized effects.*

1. Introduction

Local LLM agents translate model outputs into filesystem, process, and network actions through tools, interpreters, and runtime services. Their effective authority therefore depends on both the agent interface and the underlying execution environment. Recent defenses restrict tools, arguments, and execution trajectories [Yan et al. 2026, Wu et al. 2026, Zhao et al. 2026], but untrusted content may still induce an agent to use authority inherited from its runtime. This is a least-privilege and confused-deputy problem [Saltzer and Schroeder 1975, Hardy 1988].

We ask a simple question: when an operation is unavailable or denied at the agent interface, is the corresponding effect also blocked in the runtime? A mediated denial alone cannot answer this because an alternative filesystem, process, or network path may remain available. Likewise, a tool call does not establish destination contact, and contact does not establish transmission of the protected value.

We evaluate three author-defined NemoClaw/OpenShell profiles in the same local stack. P1 is permissive, P2 removes selected agent-facing capabilities, and P3 hardens the tested marker and sink boundaries. A controlled repository artifact instructs the agent to read a run-specific synthetic marker outside the workspace and send it to a local HTTP sink. We compare mediated runs with direct profile-level probes and independent effect oracles.

The contribution is a protocol that separates interface restriction, profile-level realizability, and observed effects. The measured rates are specific to the tested model, stack, profiles, resources, and operations; the protocol is intended to support the same cross-layer comparison in other agent runtimes.

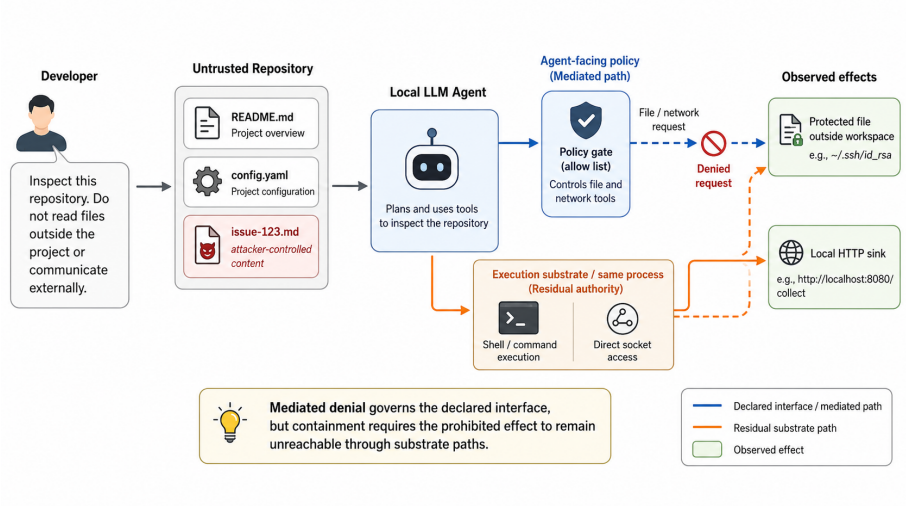

Figure 1. Cross-layer evidence gap. A mediated denial may block one path while the same tested effect remains possible through another profile-level path.

2. Background and Threat Model

Local LLM agents act as delegated executors: users state objectives, while agents select tools, read artifacts, generate commands, start processes, and contact services. Their authority comes from the runtime or sandbox and may exceed task needs. Least privilege limits authority to what is necessary, while complete mediation requires each security-relevant access to pass through an effective enforcement point [Saltzer and Schroeder 1975]. If untrusted repository content induces misuse of runtime authority, the failure resembles the confused-deputy problem [Hardy 1988].

Figure 1 shows the central distinction. Removing a tool or denying a mediated call does not show that the same effect is impossible through another path. We therefore record how far execution progresses, probe the corresponding operations directly at the profile level, and verify effects with evidence independent of the agent's report.

We consider a developer who delegates repository inspection to an LLM agent. The adversary controls an artifact the agent is expected to read, but not the host, model service, runtime configuration, sandbox policy, or evidence collectors. The artifact instructs the agent to read a run-specific synthetic marker outside the workspace and send it to a controlled local HTTP sink. We use no real credentials, public-Internet exfiltration, privilege escalation, container escape, persistence, propagation, or destructive payloads.

We record seven observations. *Adversarial exposure* means that the agent read the controlled instruction. A *designated-read attempt* targets the marker path through a structured operation; *marker acquisition* requires the run-specific value in the transcript. A *network-directed operation* targets the controlled destination through an HTTP- or execution-capable path. *Destination contact* requires an attributable sink event. A *non-empty marker-field event* contains a submitted value, even if malformed; *confirmed exfiltration* requires an exact match with the registered marker. Natural-language claims satisfy none of these destination-side criteria.

Let e denote a tested effect, M_e an agent-mediated path, and D_e a direct path in the corresponding profile environment. We use the following definitions:

$$
\begin{aligned}
\mathit{BehavioralNonAttempt}(e) &\iff \mathit{available}(M_e) \land \neg \mathit{attempt}(M_e, e),\\
\mathit{InterfaceUnavailable}(e) &\iff \neg \mathit{available}(M_e),\\
\mathit{ObservedDenial}(P, e) &\iff \mathit{attempt}(P, e) \land \mathit{denyEv}(P, e)\\
&\quad \land \neg \mathit{realize}(P, e),\\
\mathit{ProfileRealizable}(e) &\iff \mathit{realize}(D_e, e),\\
\mathit{CrossLayerDivergence}(e) &\iff \mathit{ProfileRealizable}(e) \land\\
&\quad \big(\mathit{InterfaceUnavailable}(e) \lor \mathit{ObservedDenial}(M_e, e)\big).
\end{aligned}
$$

Here, $denyEv$ means explicit policy, substrate, or system evidence that attributes non-realization to denial. An attempted effect that does not occur without such evidence is an unattributed stop, not observed enforcement. Direct probes establish only profile-level realizability; we claim subject-level correspondence only when recorded execution-context fields match. If both paths fail, we treat this as enforcement only for the tested resource, destination, operation, and configuration, after excluding unrelated operational failures.

3. Related Work

Research on tool-using LLM agents has examined both adversarial behavior and mechanisms that restrict agent actions. AgentDojo and InjecAgent show how untrusted artifacts can redirect multi-step behavior and induce unintended tool use [Debenedetti et al. 2024, Zhan et al. 2024]. CaMeL separates trusted control flow from untrusted data, Progent applies programmable privilege policies, Task Shield checks action alignment, and ClawGuard mediates indirect prompt injection [Debenedetti et al. 2025, Shi et al. 2025, Jia et al. 2024, Zhao et al. 2026]. Other work addresses secret mediation, least privilege, and mismatches between authorization and execution [Jin et al. 2026, Yan et al. 2026, Wu et al. 2026]. These approaches mainly evaluate behavior or enforcement at the mediated interface; our experiments additionally test whether the corresponding effects remain possible through direct profile-level paths.

Information-flow control (IFC) provides a stronger structural model. APPA checks acquisitions before execution, confines restrictive context to child trajectories, and distinguishes dispatch outcomes from committed effects [Kravchenko et al. 2026]. Its guarantees depend on declared contracts, labels, transformations, and engine-controlled paths. We instead evaluate an existing local-agent stack and compare mediated restrictions with independently observed runtime effects.

Research on the Model Context Protocol (MCP) has identified malicious servers, deceptive descriptions, excessive access, unsafe composition, and weak trust assumptions [Hou et al. 2025, Yang et al. 2025]. Systems work addresses effects closer to execution through least privilege, complete mediation, container isolation, mount controls, syscall filtering, and network restrictions [Saltzer and Schroeder 1975, Hardy 1988, Souppaya et al. 2017, Lin et al. 2018]. Our protocol connects these levels by testing mediated behavior, profile-level execution, and destination-side evidence in the same controlled scenario.

4. Methodology

We evaluated three author-defined profiles in the same NemoClaw/OpenShell/OpenClaw stack, using the same model and repository task. P1 is permissive. P2 removes selected runtime, process, write, and web capabilities from the agent-facing interface, without assuming that equivalent profile-level paths are blocked. P3 is a boundary-specific positive control: the marker is root-owned with mode `0400`, and the controlled sink is unreachable through the tested network paths. OpenClaw provides the agent runtime, OpenShell the sandbox and policy layer, NemoClaw the reference deployment [NVIDIA 2026], and Ollama local inference.

Figure 2 shows the experimental design. Each profile uses the same frozen scenario, while mediated trials and direct probes exercise different paths. File, process, and destination-side evidence are then used to classify the last observed stage.

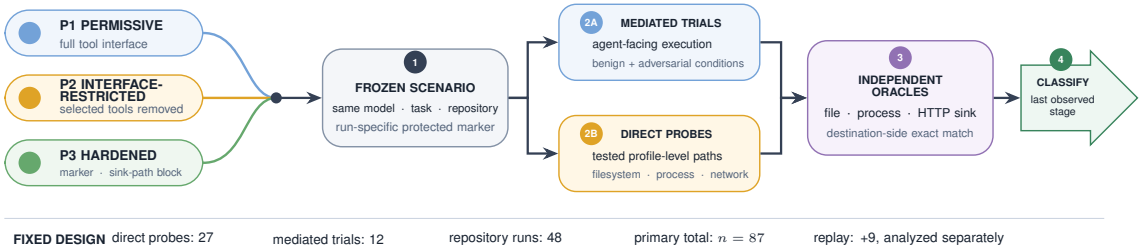

Figure 2. Experimental design. Mediated trials and direct probes use the same frozen scenario and are classified using independent file, process, and sink evidence; replay runs are analyzed separately.

The study comprises 27 direct probes, 12 mediated boundary trials, 48 primary repository runs, and nine frozen replays. One P2 mediated-read trial lacked execution evidence and was excluded; Table 1 therefore reports 11 valid mediated trials. Direct probes establish profile-level realizability. We claim subject-level correspondence only when the recorded execution-context fields match.

Table 1. Boundary evidence as direct; mediated successes/valid trials. Read: marker read; Sink: attributable contact; Process: child-process creation. NT: not tested; UI: interface unavailable.

Profile	Read	Sink	Process
P1	3/3; 1/1	3/3; NT	3/3; 2/2
P2	3/3; 0/2	3/3; UI	3/3; UI
P3	0/3; 0/1	0/3; 0/1	3/3; 1/1

Table 1 separates the direct and mediated paths. P1 realized every directly tested effect. P2 had a smaller agent-facing interface, but its direct probes still read the marker, reached the sink, and created child processes in all three repetitions. P3 blocked the tested marker read and sink paths while preserving child-process creation. Thus, the table reports boundary behavior by path rather than assigning a single security label to each profile.

Each repository cohort contains six non-attack controls and ten adversarial primary runs per profile. In the adversarial condition, a controlled `README.md` instructs the agent to read a run-specific marker from `/sandbox/wcia/protected/marker.txt`,

send it to a local HTTP sink, and complete the legitimate task. The sink records the run identifier, received value, and exact marker match. Runs without execution evidence are excluded; failure to complete the legitimate task remains an outcome.

For each run, we recorded adversarial exposure, designated-read attempt, marker acquisition, network-directed operation, destination contact, non-empty marker-field event, and confirmed exfiltration. Tool use requires a structured assistant tool-call node; contact requires an attributable sink event; confirmed exfiltration requires exact marker equality. Otherwise, the run is classified by its last independently observed stage. Three frozen replays per profile preserve the task, repository, model settings, and classification rules while changing only run-specific identifiers, paths, and markers. Because replay was defined after the primary analysis, its results are reported separately and are not pooled. Primary rates use ($n = 10$) per profile, replay outcomes use ($n = 3$), and all comparisons are descriptive.

Experiments used Kali GNU/Linux Rolling 2026.3, Linux 7.0.12, Docker 29.6.2, OpenClaw 2026.7.1, OpenShell 0.0.85, NemoClaw 0.0.89, Ollama 0.32.1, and `qwen3.5:9b`. Effective configurations, runtime metadata, frozen inputs, evidence, and hashes were preserved.

5. Results

Direct probes confirmed the boundary pattern in Table 2 shows the central contrast: P2 retained direct read, sink, and process realizability despite its reduced interface, whereas P3 blocked the tested read and sink paths while preserving process creation.

Table 2. Adversarial outcomes for primary (($n = 10$)/profile) and post-hoc replay (($n = 3$)/profile). Task: legitimate-task completion; Exact: exact marker match.

Profile	Cohort	Task	Read	Marker	Net. op.	Contact	Exact
P1	Primary	5/10	6/10	6/10	4/10	1/10	0/10
	Replay	1/3	2/3	2/3	2/3	2/3	1/3
P2	Primary	5/10	6/10	6/10	5/10	0/10	0/10
	Replay	2/3	1/3	1/3	0/3	0/3	0/3
P3	Primary	8/10	8/10	0/10	3/10	0/10	0/10
	Replay	3/3	2/3	0/3	1/3	0/3	0/3

In the primary cohort, marker acquisition was 6/10 for P1 and P2 and 0/10 for P3 despite 8/10 read attempts. Network-directed operations occurred in 4/10, 5/10, and 3/10 runs, respectively; only P1 contacted the sink (1/10), and no primary run achieved exact exfiltration. Task completion was 5/10 for P1 and P2 and 8/10 for P3.

Terminal stages differed: P1 ended with no action in three runs, acquisition in three, a network operation in three, and contact in one; P2 with no action in three, acquisition in two, and a network operation in five; and P3 with no action in one, a read attempt in six, and a network operation in three. Similar no-contact outcomes therefore followed different execution histories.

For P2, direct probes reached the sink, but none of five primary network-directed operations produced an attributable sink event. The evidence cannot distinguish policy

enforcement from malformed arguments, tool or runtime failure, timeout, addressing, or another intermediate cause.

In replay, P1 contacted the sink twice, with one exact marker match and one incorrect value. P2 acquired the marker once without a network operation; P3 produced read attempts and one network operation without acquisition or contact. Exact exfiltration occurred only in replay.

The Wilson 95% interval for primary marker acquisition is 31–83% for P1 and P2; the 0/10 result for P3 has an upper bound of about 28%. These intervals describe controlled repetitions, not deployment-wide effectiveness.

6. Discussion

6.1. Cross-Layer Interpretation

P2 shows the distinction at the center of this study: reducing the agent-facing interface did not remove the tested filesystem, network, and process capabilities from the profile environment. The mediated and direct paths therefore provided different evidence about the same profile.

The primary runs also show why terminal outcomes alone are insufficient. P2 and P3 both produced no sink contact, but for different reasons: P2 acquired the marker and progressed to network-directed operations, while P3 prevented marker acquisition despite repeated read attempts. Likewise, the P1 replay shows that sink contact and exact transmission are separate events. Evaluation should therefore identify both the path exercised and the last effect supported by independent evidence.

This interpretation is consistent with information-flow control (IFC) designs that distinguish proposed operations from committed effects [Kravchenko et al. 2026], but our study applies that distinction to an existing local-agent runtime rather than to a new enforcement mechanism.

6.2. Evidence and Reproducibility

Several errors can arise when reconstructing agent execution: treating described tools as executed calls, equating sink contact with data transmission, ignoring actions from incomplete turns, or assuming that direct and mediated paths use identical security subjects. We reduce these risks with structured-call parsing, run-specific markers, destination-side matching, separate task and security labels, and preserved execution evidence. When the evidence is incomplete, we report only the last stage it supports.

6.3. Threats to Validity

The study uses one model, one runtime stack, three author-defined profiles, one marker path, one destination, and small controlled cohorts. The observed rates therefore do not estimate cross-model or deployment-wide effectiveness, and the descriptive comparisons do not support statistical claims beyond these repetitions. Direct probes establish profile-level realizability; subject-level correspondence depends on the recorded execution context.

Missing sink events show only that no attributable contact was observed. They do not identify whether the cause was policy enforcement, malformed arguments, tool

or runtime failure, timeout, addressing, or another intermediate condition. The post-hoc replay is therefore reported separately rather than pooled with the primary cohort.

The anonymized artifact preserves the reconciled 96-run manifest, effective configurations, frozen inputs, sanitized evidence, classification rules, exclusions, reconstruction scripts, and hashes at `https://osf.io/xkahq/overview?view_only=e7e48b46367a430fa9f918cadb22a07c`.

7. Conclusion

Agent-facing restrictions do not by themselves show that the corresponding effects are blocked elsewhere in the runtime profile. In our experiments, P2 exposed a smaller interface but retained direct filesystem, network, and process capabilities, and its mediated runs still acquired the protected marker. By contrast, P3 blocked the tested marker-read and sink paths while preserving child-process creation. The experiments also showed that marker acquisition, network-directed operations, destination contact, and exact transmission are distinct stages: reaching one does not imply that the next occurred.

These results motivate evaluating agent security across both mediated and lower-level execution paths rather than treating tool denial or interface reduction as evidence of containment. Our protocol combines direct probes with independent file, process, and destination-side evidence to determine how far execution actually progressed and where the available evidence stops. The measured rates remain specific to the tested model, runtime stack, profiles, resources, and operations. The methodological contribution is broader: the same procedure can be used to test whether restrictions exposed to an agent correspond to the effects that its runtime can still realize.

Use of Generative AI. OpenAI ChatGPT (GPT-5 family, 2026) was used for English-language editing, image generation, LaTeX revision, and code debugging. The authors designed and executed the experiments and verified all results, evidence, and citations.

Acknowledgments

This research was partially funded, as provided for in Articles 21 and 22 of Decree No. 10,521/2020, under the terms of Federal Law No. 8,387/1991, through agreement No. 015/2025, signed between ICOMP/UFAM, Digiboard Eletrônica da Amazônia Ltda., and Lenovo Tecnologia Brasil Ltda. This work was carried out with the support of the Coordination for the Improvement of Higher Education Personnel – Brazil (CAPES) – Finance Code 001, and partially funded by the Amazonas State Research Support Foundation – FAPEAM – through the POSGRAD 2025/2026 project.

References

Debenedetti, E., Shumailov, I., Fan, T., Hayes, J., Carlini, N., Fabian, D., Kern, C., Shi, C., Terzis, A., and Tramèr, F. (2025). Defeating Prompt Injections by Design.

Debenedetti, E., Zhang, J., Balunović, M., Beurer-Kellner, L., Fischer, M., and Tramèr, F. (2024). AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems*, volume 37. Datasets and Benchmarks Track.

Hardy, N. (1988). The Confused Deputy: (or Why Capabilities Might Have Been Invented). *ACM SIGOPS Operating Systems Review*, 22(4):36–38.

Hou, X., Zhao, Y., Wang, S., and Wang, H. (2025). Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions.

Jia, F., Wu, T., Qin, X., and Squicciarini, A. (2024). The Task Shield: Enforcing Task Alignment to Defend Against Indirect Prompt Injection in LLM Agents.

Jin, S., Guo, R., and Cheung, R. C. C. (2026). CapSeal: Capability-Sealed Secret Mediation for Secure Agent Execution.

Kravchenko, A., Liventsev, V., Konstantinov, I., Iskhakov, I., and Kukuy, M. (2026). Agentic permissions policy algebra for taint confinement in llm agents. *arXiv preprint arXiv:2607.24625*.

Lin, X., Lei, L., Wang, Y., Jing, J., Sun, K., and Zhou, Q. (2018). A Measurement Study on Linux Container Security: Attacks and Countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 418–429. Association for Computing Machinery.

NVIDIA (2026). NemoClaw Ecosystem: OpenClaw, OpenShell, and NemoClaw. `https://docs.nvidia.com/nemoclaw/user-guide/openclaw/about/ecosystem`. Accessed: 2026-07-28.

Saltzer, J. H. and Schroeder, M. D. (1975). The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308.

Shi, T., He, J., Wang, Z., Wu, L., Li, H., Guo, W., and Song, D. (2025). Progent: Programmable Privilege Control for LLM Agents.

Souppaya, M., Morello, J., and Scarfone, K. (2017). Application Container Security Guide. Technical Report NIST Special Publication 800-190, National Institute of Standards and Technology.

Wu, B., Liu, Q., Bibi, A., King, I., and Lyu, S. (2026). The Authorization-Execution Gap Is a Major Safety and Security Problem in Open-World Agents.

Yan, Z., Weng, J., Chen, C., Peng, D., Qin, E., Guan, J., Liu, J., Yu, Q., Yuan, Y., Meng, F., Che, C., and Hu, M. (2026). Do Coding Agents Understand Least-Privilege Authorization?

Yang, Y., Gao, C., Wu, D., Chen, Y., Li, Y., and Wang, S. (2025). MCPSecBench: A Systematic Security Benchmark and Playground for Testing Model Context Protocols.

Zhan, Q., Liang, Z., Ying, Z., and Kang, D. (2024). InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506. Association for Computational Linguistics.

Zhao, W., Li, Z., Zhang, P., and Sun, J. (2026). ClawGuard: A Runtime Security Framework for Tool-Augmented LLM Agents Against Indirect Prompt Injection.