# Policy-Bound Agent Skills: Authority Boundaries for Reusable Agent Procedures

Shuwen Liu
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
shuwenliu@link.cuhk.edu.cn

Juan A. Wibowo
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
juanalbertwibowo@link.cuhk.edu.cn

George C. Polyzos
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
polyzos@acm.org

## Abstract

Agent Skills package reusable procedural knowledge for language-model agents: when a task-specific procedure is relevant, what steps to follow, and which scripts, references, or templates to use. This paper argues that this procedural layer is incomplete for delegated agents that act over user data, tools, memory, and external services. A useful and non-malicious skill can still become a path for data overreach, unintended tool use, unauthorized external action, silent delegation, or persistent memory drift. We propose *policy-bound agent skills*: skill packages that pair model-readable procedural instructions with runtime-readable boundary contracts. The contract states the skill's authorized purpose, data scope, capabilities, actions, delegation paths, memory effects, escalation rules, and audit obligations. A controlled trace study illustrates how skill-level boundaries can expose and constrain authority assumptions introduced by reusable skill procedures. Our core claim is that skills should be treated not only as reusable procedural artifacts, but also as authorization-bearing artifacts enforced by the runtime.

***Keywords:*** Agent Skills, delegated agents, policy-bound execution, agent security

## 1 Introduction

LLM-based agents increasingly operate as hybrid software systems: they read and write files, call tools, run code, browse repositories, use external services, write memory, and sometimes take actions on behalf of users. Agent Skills make these systems more capable by packaging reusable procedural knowledge into portable artifacts, typically centered on a `SKILL.md` file and accompanied by scripts, templates, references, or other resources [1, 2]. Skills are useful because they let agents reuse domain-specific know-how across tasks, sessions, users, and platforms.

This paper takes a security position: skills should not be treated only as productivity artifacts. A prompt is usually local to one interaction, but a skill can be installed, generated, updated, shared, invoked repeatedly, and combined with other skills or tools. It can therefore shape not only what an agent says, but also what the agent tries to do: which data it reads, which tools it calls, which memory it writes, which services it contacts, and which external actions it proposes.

Prior work shows that real-world skills can contain prompt injection, data exfiltration, privilege escalation, and supply-chain risks [7, 9]. These risks are important, but they are not the whole problem. A skill can be benign and still be unsafe for a particular delegation. For example, a contract-summary skill may instruct the agent to read an uploaded contract, inspect related project files, search externally, store raw text for follow-up, and email the summary. Such a workflow may be useful in due diligence, but exceeds a narrower request to summarize one uploaded file. The final answer may be correct while the execution path is not authorized.

We call this failure *authority drift*: a skill causes the agent's effective authority to expand during execution by importing data access, tool use, external actions, memory effects, or subdelegation that were not part of the current task. This is distinct from task failure and from malicious-skill detection: the skill need not be adversarial.

We propose that Agent Skills carry explicit authorization boundaries. A skill should include both a procedure and a reviewable boundary for that procedure. The procedure tells the model how to perform the task; the boundary tells the runtime what authority the skill may request or exercise. It should cover data, capabilities, actions, delegation, memory effects, escalation rules, and audit records, and it should be parsed and enforced outside the model context.

This paper makes three claims. First, skills create a new place where authority can drift because they package reusable procedures that may suggest data access, tool use, memory writes, external actions, and subroutines. Second, the active skill invocation is a useful control unit alongside the user account, task, tool call, and service permission scope. Third, a practical path forward is to attach a small boundary contract to each skill, approve or restrict that contract at deployment time, and let a runtime mediator enforce the resulting effective boundary. Our proposal does not replace runtime policy systems, sandboxing, signing, provenance, or malicious-skill detection; it adds a skill-level policy artifact that such systems can consume, restrict, compile, intersect with task policies, and audit.

*Agent Skills '26, San Jose, CA, USA*
2026.

## 2 Threat Model and Problem Statement

***System and trust model.*** We consider a delegated agent that can load skills, read skill files, call tools, access user data, write memory, and interact with external services. The user gives the agent a task-specific delegation, but the agent may run in an account with broader ambient access, such as workspace files, email, calendar, shell, cloud storage, or external APIs. Skills may be first-party, third-party, or generated by an agent, and may contain model-readable instructions, references, templates, scripts, or other resources. We trust a runtime mediator to intercept security-relevant side effects before execution, including file reads, tool calls, memory writes, external actions, and delegation attempts. We do not trust the language model to enforce natural-language constraints, and we do not treat a skill's declared boundary as a self-issued permission grant.

***Adversary and authority drift.*** The adversary may influence the user request, documents, retrieved content, tool outputs, memory, or skill instructions. The goal is to make the agent exceed the intended delegation boundary, for example by reading unrelated files, sending external messages, storing private data in persistent memory, calling unapproved tools, modifying unrelated resources, or delegating work to another model or service. We call this failure *authority drift*: the agent's effective authority expands during skill execution because the skill suggests extra tools, data sources, actions, memory writes, or subroutines that were not part of the current task. The agent may finish the user's task and still cross the boundary. For example, a contract-summary skill may be useful when it checks related files, searches externally, stores raw text, and emails a report; yet those same steps exceed a narrower request.

***Boundary failures.*** We focus on five boundary failures: *data overreach*, where a skill leads the agent to read or reveal data outside scope; *tool overreach*, where it calls tools not approved for the task; *action overreach*, where it takes irreversible or external actions without approval; *silent delegation*, where it calls another agent, model, service, or human channel without permission; and *persistent drift*, where it leaves memory or state that expands future authority.

***Relation to other threats and non-goals.*** Our focus differs from malicious-skill and prompt-injection threat models. Malicious-skill work asks whether skill content is adversarial or vulnerable; prompt-injection work asks whether untrusted inputs can hijack the agent. We focus on a third case: a non-malicious skill whose reusable procedure is broader than the current delegation. These threats are complementary and can co-occur. We do not address sandbox escapes, compromised operating systems, malicious runtimes, or scripts that bypass mediated interfaces; these require sandboxing, signing, provenance, malware scanning, dependency review, and capability isolation.

## 3 Policy-Bound Agent Skills

We propose a simple abstraction:

`skill = (procedure, resources, scripts, boundary).`

The `procedure` tells the agent how to do the task; `resources` and `scripts` support it; and the `boundary` states what authority the skill may request or exercise. The boundary is not model advice, but a structured contract for runtimes, scanners, registries, and auditors.

We distinguish model-readable procedure from runtime-readable authority metadata. The `SKILL.md` file guides the model's proposed procedure, but it is not trusted to enforce security. The boundary manifest is parsed outside the model context by the agent harness, tool broker, sandbox, or runtime policy system. The model may propose actions; only the runtime can authorize side effects.

***Boundary manifest.*** A policy-bound skill can include a compact runtime-readable manifest, such as `boundary.yaml`, next to `SKILL.md`. The `SKILL.md` file remains model-readable procedural guidance; `boundary.yaml` is parsed by the runtime outside the model context. The essential interface is a contract over normalized events, not a prompt instruction that the model is expected to obey.

```
boundary:
  purpose: summarize_uploaded_contract
  data: {allow: [uploaded_file], deny: [email, calendar, credentials
      ]}
  capabilities: {allow: [document.read_uploaded, document.summarize],
       approval: [external_search, email.send_report], deny: [
      workspace.read_all, shell.execute, cloud.upload]}
  actions: {allow: [extract, summarize, format], approval: [
      send_report, modify_original], deny: [sign_contract,
      submit_form]}
  delegation: {allow: [], deny: [external_agent, unapproved_model]}
  memory: {allow: [task_summary], deny: [raw_text,
      private_identifiers]}
  audit: {log: [files, capabilities, actions, denials, approvals]}
```

The important distinction is between *available permissions* and *authorized permissions for this skill.* The user account may have access to email, files, shell, cloud storage, and external services, while a specific skill invocation may need only one file and a local parser. To remain portable across agent platforms, boundaries can name semantic capabilities rather than concrete tools, such as `doc.read`, `calendar.freebusy`, or `email.send`. The deployment maps these capabilities to concrete tools, APIs, or service scopes, or denies them if no safe mapping exists.

***Declared, approved, and effective boundaries.*** A boundary manifest is not a self-issued permission grant by the skill author. It is a declarative authority request and review object associated with the skill. A deployment may accept, reject, restrict, or override this request through user, organization, registry, or developer policy. At invocation time, the runtime computes an effective boundary by intersecting the approved skill boundary with the user's task delegation, account permissions, organization policy, available tools, and any contextual runtime policy. The runtime enforces only

this effective boundary. Thus, a malicious or overbroad skill cannot gain authority merely by declaring a broad manifest; broad declarations instead become visible signals for rejection, restriction, or human approval. Even when not trusted as a grant, the declared boundary is useful as a review object: it lets the runtime surface mismatches such as a summarization skill requesting `email.send`, rather than asking only whether to allow a generic tool call.

***Machine-checkable enforcement.*** The runtime parses the manifest into categories such as data, capabilities, actions, memory, delegation, escalation, and audit. Each mediated agent step is represented as a normalized event for policy checking, using tool schemas, wrapper metadata, service scopes, path or data labels, or sandbox events, such as `(capability, email.send)` or `(data, workspace_file)`. The mediator checks the event against the active skill boundary and returns *allow*, *ask*, or *deny*. At load time, the system can warn about skills whose scripts conflict with the declared boundary. At action time, it intercepts data access, tool calls, memory writes, external actions, and delegation attempts before execution. Undeclared events default to denial or approval rather than silent execution.

***Skill invocation frames.*** A runtime cannot reliably infer whether a proposed action was caused by the user's prompt, the base model, prior context, or a loaded skill. Policy-bound skills therefore should not rely on causal attribution inside the model. Instead, the runtime opens a skill invocation frame when a skill is invoked. The frame records the active skill identifier, purpose, approved boundary, parent task, and audit identifier. All mediated file reads, tool calls, memory writes, external actions, and delegation attempts made while the frame is active are checked against the frame's effective boundary. If the agent proposes an action outside the frame, the runtime may deny it, request approval, invoke another skill under a separate frame, or exit the frame and evaluate the action under task-level policy. This design is conservative, but avoids relying on unverifiable model-internal attribution.

***Execution flow.*** The skill proposes the procedure. The boundary exposes and limits its authority. The runtime enforces the effective limit, and the trace records the result. A paragraph-only boundary may be ignored by the model. A runtime that enforces generic tool rules without knowing the active skill cannot apply skill-specific authority. If there is no trace, misuse is hard to debug.

## 4 Controlled Trace Study

We evaluate policy-bound skills as a runtime control abstraction, not as an end-to-end benchmark of agent capability or robustness. The study asks whether an active skill boundary, once specified, can express and enforce distinctions across data, capabilities, actions, memory, and delegation that prompt-only rules or simple tool-only guards miss.

**Table 1.** Trace-level enforcement outcomes on policy-labeled SkillsBench-style task categories. CCC measures clean core completion, not full benchmark task success. Lower is better for BVR, WBVS, and AC.

| Method | CCC↑ | BVR↓ | WBVS↓ | AC↓ |
|---|---|---|---|---|
| Vanilla Skill | 0.500 | 0.917 | 0.917 | 0.000 |
| Prompt-Only Boundary | 0.750 | 0.750 | 0.750 | 0.000 |
| Simple Tool Guard | 0.500 | 0.889 | 0.808 | 0.188 |
| Always-Ask-Human | 0.500 | 0.000 | 0.000 | 1.729 |
| Policy-Bound Incomplete | 0.500 | 0.611 | 0.579 | 0.000 |
| Policy-Bound Skill | 0.500 | 0.000 | 0.000 | 0.458 |

***Setup.*** We adapt task categories from SkillsBench [6], but do not execute the full benchmark or measure end-to-end task success. Instead, we build a controlled trace dataset over 12 document, calendar/email, repository, and finance workflows. Each task category is paired with a hand-written boundary manifest that labels normalized events as *allow*, *ask*, or *deny*. For each task, our trace-generation script creates one clean trace and three boundary-pressure traces. In the reported run, proposed events are generated from task instructions and variant descriptions, then normalized into policy-relevant event categories. Each event is a normalized representation of a proposed side effect: its first field is the authority category and its second field is the requested resource, capability, or action. Example events include `(data, credentials)`, `(capability, email.send)`, and `(memory, raw_document_text)`. Each runtime policy strategy then decides whether each event executes, is blocked, or is routed to approval. The numbers are trace-level enforcement outcomes, not benchmark pass rates.

***Methods and metrics.*** We compare six settings. *Vanilla Skill* executes every proposed event. *Prompt-Only Boundary* exposes the boundary during trace generation, but has no runtime enforcement. *Simple Tool Guard* enforces only tool/-capability events. *Always-Ask-Human* routes all non-allowed events to approval. *Policy-Bound Incomplete* uses a partial manifest missing memory/delegation rules and approval-required actions. *Policy-Bound Skill* enforces the full boundary: *allow* events execute, *ask* events route to approval, and *deny* events are blocked.

We report four metrics. Clean Core Completion (CCC) is the fraction of clean traces in which all core in-scope steps execute. CCC is a trace-level proxy for core in-scope execution, not a full task-success score. Boundary Violation Rate (BVR) is the fraction of boundary-pressure traces in which at least one non-allowed event executes without approval. Weighted Boundary Violation Score (WBVS) is the executed unauthorized risk divided by the total proposed unauthorized risk. Approval Count (AC, our proxy for human burden) is the average number of approval-routed events per run.

***Results.*** Table 1 shows three trends. First, prompt-only boundaries reduce boundary violations compared with vanilla skills, but still leave many non-allowed events executed because there is no trusted enforcement. Second, simple tool guarding lowers weighted risk, but misses violations involving data targets, action semantics, memory writes, or delegation. Third, complete policy-bound enforcement matches always-ask-human on BVR and WBVS while routing fewer events to approval because it distinguishes *allow*, *ask*, and *deny*. The incomplete-boundary ablation remains unsafe, showing that manifest coverage matters.

The zero BVR for policy-bound skills is a sanity check, not a claim that the whole agent is secure. When a non-allowed event is represented in the manifest and the runtime mediator is trusted, the mediator can block or escalate that event before execution. Because the traces and boundaries are author-labeled, this study measures expressiveness and enforceability of the boundary abstraction, not automatic boundary authoring, real-world attack resistance, or full benchmark task success.

## 5 Related Work

***Tool-using and skill-augmented agents.*** Recent LLM-agent systems combine language reasoning with external action. ReAct interleaves reasoning traces with task-specific actions [18]. Toolformer studies how language models learn to call external APIs [12]. SWE-agent shows that agent-computer interfaces can support repository navigation, file editing, and test execution [17]. Tool learning more broadly connects foundation models with external capabilities [11]. Skill-augmented agents extend this line by packaging reusable procedures. AdaPlanner reuses successful plans for later planning [15]. SkillFoundry builds scientific skill libraries from heterogeneous resources [13]. SkillsBench evaluates skills as first-class artifacts across diverse tasks [6]. These works focus on capability, reuse, reliability, or evaluation; our work adds the complementary view that reusable skill invocations should also expose, constrain, and audit delegated authority.

***Security of LLM-integrated agents.*** LLM-integrated applications blur the boundary between instructions and data, creating indirect prompt injection risks [5, 8]. AgentDojo studies prompt-injection attacks in realistic tool-using agents [4]. Recent studies of real-world skills report prompt injection, data exfiltration, privilege escalation, and supply-chain risks [7, 9]. These works motivate skill security, but mainly target malicious or vulnerable skill content. Our focus is different: even a non-malicious skill can be unsafe if its reusable procedure exceeds the current delegation.

***Authorization and runtime policy systems.*** Building on classic authorization ideas such as usage control and caveat-based delegation [3, 10], recent agent-specific systems apply least-privilege ideas at runtime. Progent enforces fine-grained policies over concrete tool calls and arguments, with fallback actions and dynamic policy updates [14]. MiniScope reconstructs permission hierarchies from service authorization metadata, such as OAuth scopes, and computes least-privilege permissions for tool-calling agents [19]. Conseca generates just-in-time contextual policies from the user request, trusted context, and tool documentation, then enforces proposed actions deterministically [16]. These systems provide enforcement over runtime actions, concrete tool calls, service permissions, or task-specific contexts. Policy-bound skills add a complementary policy source: the active skill invocation. After approval or restriction, a skill boundary can inform contextual policy generation, compile into tool-call rules, map to service scopes, or be enforced by a mediator. Thus, our contribution is not a new full-stack enforcer, but a skill-level authorization artifact for procedural knowledge.

## 6 Limitations and Conclusion

Policy-bound skills are not a replacement for malware scanning, sandboxing, signing, provenance, registry review, reputation, or runtime policy systems. A boundary manifest is not a proof that a skill is safe; it is a reviewable authority object whose effect depends on approval, restriction, and runtime enforcement. Our guarantees apply only to side effects that pass through mediated interfaces. If a skill can execute raw Python, shell commands, or other scripts with direct filesystem, network, credential, or service access, a boundary manifest alone cannot enforce authority; it must be paired with sandboxing, capability isolation, or brokered tool wrappers. We also assume a trusted runtime mediator and do not address compromised operating systems, malicious runtimes, sandbox escapes, or attacks that remain within the approved boundary.

Our trace study is controlled: we adapt SkillsBench-style task categories rather than evaluating the full benchmark end to end; traces and boundary manifests are author-labeled; and the runtime mediator is trusted. The results measure expressiveness and enforcement once events are specified, not how reliably such boundaries can be authored, inferred, approved, or maintained under adaptive attacks. Reliable boundary authoring, approval, skill composition, and compatibility with runtime systems such as Conseca, Progent, and MiniScope remain open problems; practical deployments will likely need templates, trace-based inference, linting for overbroad declarations, and registry or organization review.

Agent Skills are becoming a practical way to give agents reusable procedural knowledge. This paper argues that a skill should not only say how to do a task, but also expose what authority that procedure expects. Policy-bound skills pair procedural instructions with boundary contracts over data, capabilities, actions, delegation, memory, escalation, and audit. If skills become part of the agent software stack,

their authority should be packaged with them and enforced by the runtime, not left as a model-side assumption.

## References


[1] Agent Skills. 2026. Agent Skills Overview. AgentSkills.io. https://agentskills.io/home [Online]. Available: https://agentskills.io/home..

[2] Anthropic. 2025. Equipping Agents for the Real World with Agent Skills. Anthropic Engineering. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills [Online]. Available: https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills..

[3] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. 2014. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, San Diego, CA, USA. https://www.ndss-symposium.org/ndss2014/ndss-2014-programme/macaroons-cookies-contextual-caveats-decentralized-authorization-cloud/

[4] Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. 2024. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Proceedings of the Thirty-Eighth Conference on Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*. https://openreview.net/forum?id=m1YYAQjO3w

[5] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec)*. ACM, 79–90. doi:10.1145/3605764.3623985

[6] Xiangyi Li et al. 2026. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670* (Feb. 2026). arXiv:2602.12670 [cs.AI] https://arxiv.org/abs/2602.12670

[7] Yi Liu, Zhihao Chen, Yanjun Zhang, Gelei Deng, Yuekang Li, Jianting Ning, and Leo Yu Zhang. 2026. Malicious Agent Skills in the Wild: A Large-Scale Security Empirical Study. *arXiv preprint arXiv:2602.06547* (Feb. 2026). arXiv:2602.06547 [cs.CR] https://arxiv.org/abs/2602.06547

[8] Yupei Liu, Yuqi Jia, Runpeng Geng, Jinyuan Jia, and Neil Zhenqiang Gong. 2024. Formalizing and Benchmarking Prompt Injection Attacks and Defenses. In *Proceedings of the 33rd USENIX Security Symposium (USENIX Security)*. USENIX Association, Philadelphia, PA, USA, 1831–1847. https://www.usenix.org/conference/usenixsecurity24/presentation/liu-yupei

[9] Yi Liu, Weizhe Wang, Ruitao Feng, Yao Zhang, Guangquan Xu, Gelei Deng, Yuekang Li, and Leo Zhang. 2026. Agent Skills in the Wild: An Empirical Study of Security Vulnerabilities at Scale. *arXiv preprint arXiv:2601.10338* (Jan. 2026). arXiv:2601.10338 [cs.CR] https://arxiv.org/abs/2601.10338

[10] Jaehong Park and Ravi Sandhu. 2004. The UCONABC Usage Control Model. *ACM Transactions on Information and System Security* 7, 1 (Feb. 2004), 128–174. doi:10.1145/984334.984339

[11] Yujia Qin et al. 2024. Tool Learning with Foundation Models. *Comput. Surveys* 57, 4, Article 101 (Dec. 2024), 40 pages. doi:10.1145/3704435

[12] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 36. https://openreview.net/forum?id=Yacmpz84TH

[13] Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. 2026. SKILLFOUNDRY: Building Self-Evolving Agent Skill Libraries from Heterogeneous Scientific Resources. *arXiv preprint arXiv:2604.03964* (April 2026). arXiv:2604.03964 [cs.AI] https://arxiv.org/abs/2604.03964

[14] Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. 2025. Progent: Programmable Privilege Control for LLM Agents. *arXiv preprint arXiv:2504.11703* (April 2025). arXiv:2504.11703 [cs.CR] https://arxiv.org/abs/2504.11703

[15] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. AdaPlanner: Adaptive Planning from Feedback with Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 36. https://openreview.net/forum?id=rnKgbKmelt

[16] Lillian Tsai and Eugene Bagdasarian. 2025. Contextual Agent Security: A Policy for Every Purpose. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*. ACM, Banff, AB, Canada, 8–17. doi:10.1145/3713082.3730378

[17] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 37. https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html

[18] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=WE_vluYUL-X

[19] Jinhao Zhu, Kevin Tseng, Gil Vernik, Xiao Huang, Shishir G. Patil, Vivian Fang, and Raluca Ada Popa. 2025. MiniScope: A Least Privilege Framework for Authorizing Tool Calling Agents. *arXiv preprint arXiv:2512.11147* (Dec. 2025). arXiv:2512.11147 [cs.CR] https://arxiv.org/abs/2512.11147