
A Task Specific Gatekeeper for AI Agent Tools

Adolfo Viguera
Independent AI
Research Engineer

With
Apart Research

Abstract

AI agent evaluations may restrict networks and sandboxes without checking whether each tool action belongs to the task. This creates a gap between what an agent can execute and what it is actually authorized to do. In OpenAI's 2026 ExploitGym incident, agents exploited shared infrastructure, escaped their intended isolation, and reached Hugging Face while production safeguards were not applied to the evaluation. Bouncer adds a control step between the moment an agent generates a tool call and the moment that tool call is actually executed. It binds trusted agents and run context, checks the call against a task-specific allowlist and same-agent history, logs the decision, and executes only allowed calls. I tested it with a decide-only replay of 13,661 revisions from the Collusion Wiki case, published by Nightingale Collective and supplied for this sprint by Apart Research [5], plus controlled scripts with known effects. With operator tasks and call traces, the method could run in shadow mode and then block high-confidence out-of-task actions before they become escape paths.

1. Introduction

A sandbox or network allowlist controls where an agent can run code and where traffic can go. It does not always control what the agent does with an allowed tool or service. A package service can also carry messages, an HTTP GET can change a wiki and a permitted call can become abusive when repeated. This matters because a single out-of-task action can turn an isolated evaluation into access to shared infrastructure or an external system.

The 2026 OpenAI/Hugging Face incident shows the practical risk. OpenAI reports that agents exploited shared research infrastructure, escaped their intended isolation, and later compromised Hugging Face. The affected evaluation did not use the chain-of-thought monitors and production safeguards that OpenAI applies elsewhere [1,2]. OpenAI still had substantial traces, as METR and Redwood reviewed roughly 1,300 transcripts containing agent reasoning and actions [3]. The gap relevant here was the following. Public reports do not describe a deterministic check that compares every pending operation with the agent's task and permissions before execution.

Bouncer places that check between the tool call and its effect. Its threat model is a tool-using agent that may request an unlisted operation or misuse an allowed operation over time. The trusted harness authenticates the agent and runs, selects the policy, and routes the call through Bouncer. The agent cannot grant itself permissions by changing its arguments. Bouncer does not control tools or processes that bypass this execution path, so it complements rather than replaces sandboxing and network controls.

The main contributions are:

- 1. A task-specific enforcement point. I built a local gatekeeper and executor that apply default-deny permissions before execution, keep same-agent history, and record the decision and result outside the tool's working directory.*
- 2. A reproducible test of decisions and effects. I separate a decide-only replay of 13,661 historical wiki revisions from controlled scripts with known authorization. This shows both that blocked calls do not reach their tools and that the history rule can obstruct legitimate work under a broken clock.*
- 3. A path to use the method in the OpenAI/Hugging Face setting. We compare potential task-level and egress visibility across nine published attack phases and specify the operator data and staged evaluation needed before making a historical prevention claim.*

2. Related Work

The closest prior work falls into two groups: controls built into agent evaluations, and systems that inspect tool calls themselves.

ExploitGym belongs to the first group. Version 1.1, dated 18 June 2026, separates installation and execution traffic, removes the agent container's default internet route, and limits the execution proxy to four model API domains. It also blocks known forms of provider-hosted retrieval [6]. Despite being meaningful controls, they mainly restrict destinations and known retrieval mechanisms and they do not describe every operation that each task authorizes. Moreover, although version 1.1 was public before the July incident, I do not know which configuration OpenAI actually deployed.

AgentDojo, meanwhile, provides realistic tasks for testing prompt-injection attacks against tool-using agents [7]. It is primarily a benchmark, not a runtime gate. ClawGuard is closer to Bouncer because it derives task-specific constraints and applies deterministic rules at the tool-call boundary [8]. ToolGuardian goes further: it vets tools before admission and reasons about their effects and composition through a logic-based policy system [9]. In other words, task-aware enforcement before execution already exists. I do not claim to have invented it or to outperform these systems.

So, where is Bouncer useful? It fits settings where the task owner can list the allowed operations and wants a small, local and deterministic baseline, with same-agent history and evidence that can be inspected afterwards. What Bouncer adds here is an incident-response perspective: it separates historical effects from complete calls, checks that a local block really prevents the effect, exposes a failure in its own timing rule, and spells out the data needed to test the method on the OpenAI/Hugging Face case.

3. Methods

3.1 The general control

I place Bouncer between a proposed tool call and its executor. The agent presents the call and an identity or credential; the trusted harness authenticates it and binds the correct run, agent and policy. An `agent\_id` inside the call arguments cannot grant broader permissions.

The decision then follows a fixed path:

- 1. `per\_call` checks the tool, operation, and available arguments against the task's allowlist. Anything absent is blocked by default.*
- 2. If the call is allowed, `per\_history` checks rules that depend on previous authorized calls from the same agent.*
- 3. The executor records the call, trusted identity, rule, reason, decision and result reference.*
- 4. The tool runs only after an allow decision. A block has no tool effect.*

The checks use fixed rules, so model review for ambiguous cases is future work.

3.2 Policy and local implementation

To test the idea, I created a local research task with timed rounds. The task is reconstructed from the public wiki report and preserved messages; it is not the original prompt given to the agents. Its YAML policy allows five calls: ``web.search``, ``page.read``, ``round.question``, ``answer.submit``, and ``clock.wait``. Anything else is denied by default.

The policy also contains one history rule, ``wait-costs-real-time``. It limits the amount of task time an agent may request through ``clock.wait`` to the wall time that has actually elapsed for that agent. Because the check must happen before the effect, the executor reserves this budget before running the tool. It then returns any unused amount after a failure or a shortened wait. Finally, the executor writes its decision log outside the tools' working directory. This keeps the log away from the scripted agent, although it does not make the record cryptographically immutable.

3.3 Two separate evaluations

The two evaluations answer different questions and therefore remain separate.

First, the historical replay asks how the fixed policy treats observed wiki edits. It uses Nightingale Collective's frozen download [5], supplied for the sprint by Apart Research. Of 14,591 saved revisions, I exclude 31 attributed to known human handles and 899 without an identity. This leaves 13,661 revisions from 3,099 literal IDs. The names are useful identifiers, but they are not authenticated identities.

For replay, I explicitly adapt each saved revision to ``wiki.edit``, as the dataset lacks the original task and complete calls. Ideally, I would have the original list of tools the agent was allowed to use, and anything outside that list would be considered unauthorized.

Second, the controlled evaluation tests known legitimate work and verifies blocked effects. My scripts cover normal work, a forbidden edit, an unknown tool, chained waits and two agents with separate histories. I compare ``per_call`` with ``per_call+per_history`` using matched clocks and a broken clock that advances task time faster than wall time. These are mechanical tests, not tests of model behaviour

3.4 Applying Bouncer to high-risk agent evaluations

The wiki is only a worked example of a more general method. To evaluate that method on the OpenAI/Hugging Face case, I would need data that only the operator can provide, such as the original task and permissions, authenticated run and agent identities, ordered calls with their arguments and times, the resulting decisions and tool outputs, links to external effects, permissions for shared resources, and independently reviewed examples.

I would begin with an offline replay to measure both useful coverage and incorrect blocks. Next, I would run Bouncer in shadow mode: it would record decisions, but it would not yet change execution. Only after those checks would I allow high-confidence rules to block real calls.

4. Results

Table 1 reports the historical replay and controlled execution separately. The replay contains observed effects but no original tool calls or labels; the controlled scripts have known authorization and observable tool effects.

Table 1. Controlled tests: what worked and what failed

Test	Per-call only	+ same-agent history	Main finding
Normal task, matched clock	12/12 allowed Task completed	12/12 allowed Task completed	No legitimate call was blocked in this setup.
Forbidden wiki edit	Blocked No effect	Blocked No effect	The pre-execution block worked; the wiki did not change.
Normal task, broken clock	12/12 allowed Task completed	11/12 allowed Task stopped	History blocked a legitimate call: a false positive.
Repeated waits, broken clock	5/5 allowed	3/5 allowed	History blocked two repeated waits, as intended.
Two agents, shared clock	4/4 allowed	3/4 allowed	Per-agent history did not protect the shared clock.

Note. These are scripted mechanism tests, not model trials or population-level error rates.

5. Discussion and Limitations

Bouncer is a runtime control, not a framework for running dangerous evaluations. It checks whether a proposed action is allowed by the task before it is executed. This complements network and sandbox controls, which may miss internal actions or state changes at allowed destinations.

In an OpenAI-like evaluation, Bouncer could interrupt an escape if every relevant action passed through it and the policy rejected the first out-of-task step. I cannot claim it would have prevented the 2026 incident: the original calls and policies are unavailable, and some shell and exploit work was legitimate.

Production use requires complete tool routing, current policies, safe handling of missing context, atomic state for concurrent calls, and protected remote logs. The reported timings exclude integration, policy authoring, exceptions, storage and review. Finally, historical data contain effects rather than full trajectories, while controlled scripts contain no model behaviour

To sum up, the goal of this approach was to identify a clear intervention point for problems like the OpenAI–Hugging Face incident: the moment between a model generating a tool call and that tool call being executed.

Future Work

From here, I see five steps before this could become a usable control:

- 1. Operator replay: test reviewed task policies against real trajectories and measure coverage and legitimate blocks.*
- 2. Shadow mode: inspect live calls and record decisions without blocking them.*
- 3. Narrow enforcement: block only high-confidence violations and measure exceptions, latency, and policy cost.*
- 4. Multi-agent enforcement: add atomic state, run-level rules for shared resources, and tamper-evident remote logs.*
- 5. Ambiguous cases: compare model-assisted review (classification) and cross-agent analysis with the deterministic baseline. Hard rules must keep priority.*

Each step should only move forward if the results justify it, and there should be a way to roll back if needed. I would only add more complexity if it clearly improves the results

6. Conclusion

Bouncer makes an agent's assigned task enforceable at runtime. The prototype blocks unlisted operations before their effects and records auditable decisions; the wiki study tests those mechanics. With complete tool routing and operator data, the same pattern could support coding, research, cloud and high-risk evaluation agents. History-based and multi-agent rules still need further work.

Code and Data

Include links if applicable. If your project doesn't involve code (e.g., policy analysis) or if there are info-hazard considerations, note that here.

- **Code repository:** (<https://github.com/adoviguera/bouncer>). It contains the gatekeeper, local executor, task policy, controlled scenarios, tests and reproduction instructions.*
- **Data/Dataset:** <https://collusion.wiki> published by Nightingale Collective and supplied for this sprint. The repository stores the prepared replay data with its cleaning record separately.*

References

Use a consistent citation format. Include: Author(s), Year, Title, Venue/Publisher, and URL or DOI where available.

1. OpenAI. (2026). *The Hugging Face incident and the road ahead*.
<https://openai.com/index/hugging-face-incident-and-the-road-ahead/>
2. OpenAI. (2026). *OpenAI–Hugging Face incident technical report*. OpenAI.
<https://cdn.openai.com/pdf/67869394-cb91-4c12-888c-5cbd85c7814c/OpenAI-Hugging-Face%20Incident-Technical-Report.pdf>
3. Ryan Greenblatt, Ajeya Cotra, and Hjalmar Wijk. (2026). *Brief independent investigation of agents' behavior, reasoning and collaboration in the OpenAI/Hugging Face hacking incident*. METR and Redwood Research.
<https://metr.org/blog/2026-08-26-openai-hugging-face-incident-investigation/>
4. Hugo Larcher, Adrien Carreira, Raphael G., and Christophe Rannou. (2026). *Anatomy of a frontier lab agent intrusion: A technical timeline of the July 2026 incident*. Hugging Face Blog. <https://huggingface.co/blog/agent-intrusion-technical-timeline>
5. Sydney Von Arx, Cormac Slade Byrd, Spencer Kitts, and Thomas Larsen. (2026). *Discovery of a new OpenAI agent message board*. Nightingale Collective / Collusion Wiki.
<https://collusion.wiki/>
6. Zhun Wang et al. (2026). *ExploitGym: Can AI agents turn security vulnerabilities into real attacks?* arXiv preprint arXiv:2605.11086 (<https://arxiv.org/abs/2605.11086>); software and version 1.1 documentation published by Sunblaze UC Berkeley (<https://github.com/sunblaze-ucb/exploitgym/blob/main/CHANGELOG.md> <https://github.com/sunblaze-ucb/exploitgym/blob/main/docs/firewall.md>)
7. Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. (2024). *AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents*. *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*. <https://arxiv.org/abs/2406.13352>
8. Wei Zhao, Zhe Li, Peixin Zhang, and Jun Sun. (2026). *ClawGuard: A runtime security framework for tool-augmented LLM agents against indirect prompt injection*.
<https://arxiv.org/abs/2604.11790>

9. Arun Ravindran and Saurabh Deochake. (2026). ToolGuardian: Declarative security for AI agent-tool interactions. <https://arxiv.org/abs/2607.21835>
10. Adolfo Viguera. (2026). GitHub. <https://github.com/adoviguera/bouncer>

LLM Usage Statement

I used LLMs during the research and preparation stage to gather and organize the information I needed before starting the implementation. This helped me build enough context around the scenario, the requirements, and the approach I wanted to take.

For the coding itself, I mainly used Claude Code as a coding assistant throughout the implementation process.

I also used LLMs to help organize the resources and results into a structure similar to the provided template. However, this was done separately from the template and only served as a starting point. I then wrote the final project report myself.