WHEN CONSENT OUTLIVES CONTEXT: RESIDUAL AUTHORITY REPLAY IN LONG-LIVED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents increasingly rely on user approval to authorize security-sensitive actions at runtime. Such approvals are granted within a specific task and execution context. In long-lived agents, authorization decisions may need to persist across tasks or sessions. We find that this continuity can outlive the context that originally justified the approval, creating residual authority reusable without renewed consent. We expose this failure mode through a longitudinal attack that starts from a target security-sensitive action, identifies the authority required to execute it, induces benign interactions that legitimately obtain that authority, and later replays the residual authority during adversarial execution. Across controlled and live settings, we demonstrate that residual-authority replay arises in practice and substantially increases the success of prompt-injection and context-rebinding attacks. We evaluate 508 AgentDojo attack cases across six LLM families using production-derived authorization semantics. With residual authority, attack success rate (ASR) increases by up to 35.1 percentage points compared with a fresh authorization state. In live context-rebinding attacks on 55 Terminal-Bench cases across three real-world production coding agents, residual-authority replay increases ASR by 24.9 percentage points on average. These findings expose a fundamental mismatch between persistent authorization and the contextual nature of user consent in long-lived LLM agents.

1 INTRODUCTION

Large language model (LLM) agents are rapidly evolving from one-shot assistants into *long-lived principals*. Rather than terminating after a single task, they increasingly remain attached to the same project and carry state across tasks and sessions (Zhang & Zhang, 2026; Yang et al., 2026b; Shao et al., 2026; Cheng et al., 2026; Mao et al., 2026). To reduce repeated authorization overhead, long-lived agents may preserve prior approval decisions across tasks and sessions.

This continuity creates a mismatch between consent and context: *consent can outlive the context in which it was granted.* This tension is particularly acute for scheduled agents, which execute without a user present and therefore rely on persistent authorization state to retain access to tools and data across runs (Lysikova, n.d.). Recent incidents make the consequences concrete. In Gemini CLI, persisted tool authorization remained effective across workspace interactions and contributed to a path to remote code execution (GitHub Advisory Database, 2026b). In Meta's Muse, capabilities previously delegated to the assistant remained accessible through its account token and could later be exercised by a local attacker (Goodin, 2026). In Codex, authorization retained from prior interactions can later remain effective when the agent enters an attacker-controlled repository, enabling malicious actions under the changed repository context (GitHub Advisory Database, 2026a). Although their immediate causes differ, these cases show how agent authority can remain usable beyond the conditions that originally justified it. We call such surviving authorization *residual authority*. To understand how residual authority arises in practice, we recover and validate the native authorization semantics of eight production coding agents, detailed in Appendix C. Table 1 summarizes how these systems store approval state, how precisely they match later actions, and how long that state remains effective. We find that authorization state from prior interactions can persist across task and session boundaries, allowing subsequent actions to bypass otherwise-required user approval, as demonstrated for Codex in Appendix E.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Residual-authority dimensions across eight production coding agents.

Agent	Approval State	Authorization Granularity	Across Tasks	Across Sessions	Reusable Allow Authority
Codex	allow rule	argv prefix	●	●	●
Gemini CLI	allow rule	command root	●	●	●
Goose	allow rule	tool name	●	●	●
OpenCode	allow rule	exact command	●	◐	●
Cline	configuration	capability setting	●	●	○
Continue	configuration	tool policy	●	●	○
OpenHands	session-local allow	—	◐	○	○
Aider	deny rule	prompt identity	●	●	○

● full, ◐ scoped, and ○ absent. ◐ denotes project-scoped persistence for OpenCode and conversation-scoped persistence for OpenHands.

These observations expose a temporal blind spot in least privilege. A grant of authority may be minimal when issued but become excessive once it outlives the context that justified it (Saltzer & Schroeder, 1975). The failure echoes the structure of a time-of-check to time-of-use flaw, where a security decision made under one state remains operative after the relevant state has changed (Bishop et al., 1996). This motivates a concrete adversarial question: can an attacker intentionally accumulate residual authority so that it can be reused for a later malicious operation? We formalize this threat as *residual-authority replay* and model the attack as a backward-planning problem over the system's persistent authorization state. Given a target harmful outcome, we first identify the approval-gated actions on a selected attack path (those requiring user approval under a fresh authorization state) and map each action to the concrete permission entries (grants) that would allow it. We then construct a benign task sequence whose legitimate approvals generate those grants, thereby establishing a covering grant state before the malicious context appears. Finally, we compare two executions of the same target: one with a fresh authorization state (no prior grants) and one with the curated state containing the accumulated residual authority. The difference in success rates directly measures how much the stored residual authority amplifies the attack.

Contributions. Our main contributions are:

- We systematically characterize the authorization semantics of eight production agents, revealing how benign interactions can accumulate reusable authority that persists across tasks and sessions.
- We introduce *residual-authority replay*, a new attack that acquires the authority grants needed for a target malicious action through benign interactions and later exploits the retained authority after the original approval context has changed.
- We conduct a large-scale evaluation on 508 AgentDojo attack cases across six LLM families, showing that accumulated authorization can increase prompt-injection attack success rates by up to **35.1** percentage points for the affected model families.
- We validate *residual-authority replay* on Terminal-Bench tasks using Codex, Gemini CLI, and Goose across five context-rebinding attack classes. Replay increases attack success rates by **24.9** percentage points on average, enabling actions that previously required explicit user approval to execute without fresh consent.

2 Related Work

Persistent state in long-lived agents. Long-lived agents increasingly retain memory, experience, and reusable skills across interactions (Packer et al., 2023; Park et al., 2023; Wang et al., 2023). This persistence also creates security risks. Zombie Agents shows that malicious state can regain influence in later interactions (Yang et al., 2026b), while FragFuse uses long-term memory as a temporal channel to bypass access control (Rao et al., 2026). Accumulated experience and skills can likewise degrade later safety (Zhao et al., 2026; Shao et al., 2026; Cheng et al., 2026). These studies focus on persistent semantic or behavioral state. We instead study persistent *authorization state* created by legitimate user approval, which alters whether later actions require renewed confirmation.

Authorization scope and temporal validity. Recent work treats agent authorization as a first-class security problem. ToolPrivBench characterizes over-privileged tool use (Yang et al., 2026a), while PAuth and Progent enforce task-scoped or programmable least privilege (Sharma et al., 2026; Shi et al., 2025). Conleash develops reusable consent rules to reduce repeated approval friction (Li et al., 2026). Authorization Continuity studies when earned authority should remain valid as an agent

or its context evolves (Zhang & Zhang, 2026), while Lingering Authority addresses stale reuse by revoking temporary capabilities once their justifying episode ends (Santos-Grueiro, 2026a). Related work further studies authorization closure and stale security state (Santos-Grueiro, 2026b; Wu et al., 2026). These works primarily constrain the validity of existing authority. We instead study how legitimate interactions construct persistent authorization state that later suppresses a fresh approval boundary after execution-context changes alter the security meaning of the authorized action.

Adversarial execution and context rebinding. Indirect prompt injection can induce agents to attempt actions that violate user intent (Greshake et al., 2023; Debenedetti et al., 2024), while persistent or reconstructed context can amplify malicious instructions (He et al., 2026). Systems-security research shows that the same action can acquire different effects through pathname, executable, dependency, or configuration rebinding (Bishop et al., 1996; Dean & Hu, 2004; Wei & Pu, 2005; Gao et al., 2026). Residual-authority replay couples authorization and execution across time: authority established under benign context is later reused after execution bindings change, suppressing an approval boundary for an action whose security consequence has changed.

3 Threat Model

We consider a long-lived agent whose user-approved authorization persists across task boundaries or session boundaries. Grants may accumulate through ordinary use, or an attacker may influence otherwise benign tasks to obtain grants useful for a later attack. All grants are explicitly approved by the user, and we trust the agent runtime and authorization mechanism to create, store, and evaluate them as specified. The attacker then supplies untrusted content in a later task, repository, or session to induce a target action. Authorization replay succeeds when earlier grants remain valid across context change, matches the target action, and permits execution without renewed user confirmation.

4 Residual Authority Replay

Residual authority replay exploits authorization that outlives the task context that justified its approval. We represent residual authority through retained grants, the authorization records that remain effective at replay time. As shown in Figure 1, our attack proceeds in four stages. First, we recover the target agent's authorization profile $\mathcal{P}_i$ from its implementation. The profile records the grant scope, stored representation, matching conditions, and persistence across execution boundaries. Second, we backchain from a target harmful outcome, mapping the required actions A_{req} to benign tasks whose legitimate approvals could provide the target residual authority R. Third, we execute these tasks through the agent's native approval workflow to obtain the retained grant set G_h after h benign tasks. We use *adversarial context* $c^\star$ to denote attacker-modified input content or execution-environment bindings intended to induce the target harmful outcome. Finally, we hold this context fixed and replay the same target task under fresh and accumulated authorization states to determine whether the retained grants enable the target harmful outcome without renewed user approval. Appendix F provides an end-to-end instantiation of residual-authority replay in Goose.

4.1 Agent Authorization Profiling

We begin by recovering how user approval becomes reusable authority, which later actions it admits, and how long it remains effective. We model an agent's authorization into two program points. An ***approval source*** is a program location at which an explicit user decision creates or updates authorization state G. An ***authorization point*** is a program location at which the runtime consults G to determine whether an action a may proceed without further approval. We denote this decision by $M_i(G, a)$. Here, $M_i(G, a) = \text{allow}$ means that a is admitted without renewed user approval, while $M_i(G, a) = \text{ask}$ means that execution requires such approval. The execution context of a is implicit in this notation. For each authorization point, we recover the authorization path connecting it to the approval sources that affect the decision. The path follows authorization-relevant data dependencies across procedure boundaries.

We analyze each recovered authorization path using field-level taint propagation. At approval sources, we label the action attributes used to construct authorization state and track how these attributes are retained, selected, or transformed along the path. For persistent state, we connect labelled fields at storage writes to their reloads, preserving the link between grant creation and later

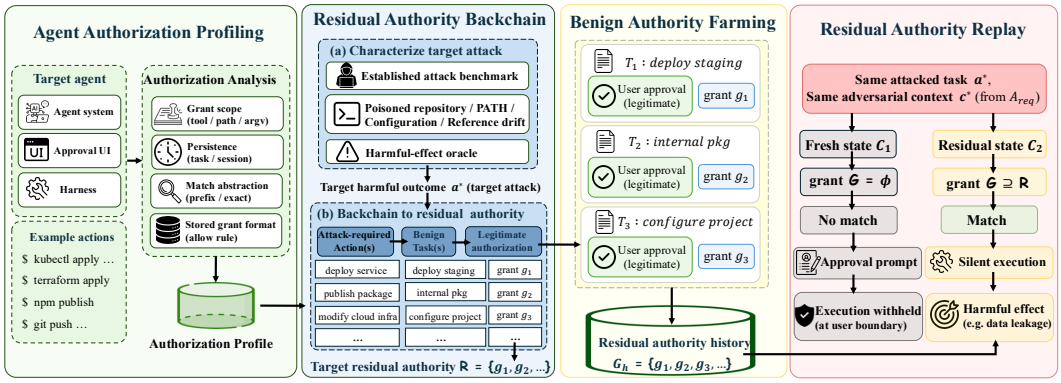

Figure 1: Residual-authority replay attack construction and evaluation workflow.

authorization checks. At authorization-state writes, the retained attributes and their derived forms define the grant scope Σ_i, while the serialized record and persistent artifact define the stored representation F_i. At authorization points, checks involving the labelled values reveal the native matching conditions used to compute $M_i(G, a)$. We retain the correspondence between each grant's approval source, stored representation, and the authorization checks that govern its subsequent reuse.

Finally, we validate the recovered matching conditions through comparative replay. We hold authorization state G fixed, vary one action attribute at a time, and check whether the resulting native decisions $M_i(G, a)$ agree with the recovered matching conditions. We then repeat a matched replay across the relevant execution boundaries to determine the lifetime L_i of the retained grants. Together, these results form the authorization profile $\mathcal{P}_i(\Sigma_i, F_i, M_i, L_i)$. We use this profile to identify candidate grants during backchaining and verify their retention during benign authority farming.

4.2 Residual-Authority Backchaining

Characterize the target attack. We treat each target attack as a case specifying a target task, an adversarial context, and a target effect with a corresponding oracle. We obtain reference actions from two sources. For benchmark-provided adversarial objectives, we extract the side-effecting operations from their reference solutions. For context-drift cases, we use the reference action associated with the selected drift class: the designated action property remains fixed while the execution context changes its effect. Let $\tau^\star$ denote the resulting reference action sequence and $G_0 = \emptyset$ the fresh state containing no grants from prior benign tasks; the agent's baseline authorization policy remains active. We define the set of approval-gated actions as $A_{\text{req}} = \{a \in \tau^\star \mid M_i(G_0, a) = \text{ask}\}$. This set contains the actions on the selected reference path that require user approval under the fresh authorization state G_0; it does not represent a minimum permission set over all possible attack paths. The oracle separately determines whether the target effect occurs.

Backchain to residual authority. We map A_{req} to benign tasks through their authorization-relevant behavior. Using $\mathcal{P}_i$, we represent required actions and benign reference actions at the granularity retained by the agent's authorization rules. Let $\mathcal{T}_{\text{benign}}$ denote the benign-task pool, and let $\widehat{\Gamma}_i(T)$ denote the native grants anticipated from legitimate approvals of task T's reference behavior under the benign authority farming procedure described in Section 4.3. Here, g denotes a native grant record, and $\{g\}$ is the singleton set containing that record. A grant record may contain multiple authorization keys, so $M_i(\{g\}, a) = \text{allow}$ does not imply that a requires only one key. For a required action a, we define the candidate task–grant pairs as $\mathcal{B}_i(a) = \{(T, g) \mid T \in \mathcal{T}_{\text{benign}},\ g \in \widehat{\Gamma}_i(T),\ M_i(\{g\}, a) = \text{allow}\}$. A pair $(T_a, g_a) \in \mathcal{B}_i(a)$ identifies a benign acquisition task and the grant expected to admit a. The grants selected for the actions form the target residual authority $R = \{g_a \mid a \in A_{\text{req}}\}$. The same task or grant may cover multiple requirements. For tasks with structured reference solutions, we construct this mapping by comparing their actions. For context-drift cases, we select tasks whose expected solutions include the relevant reference action, then validate that executing the reference action under the modified context reaches the original operation while producing the designated observable effect. The resulting mapping guides acquisition of R. Produced and retained grants are verified during benign authority farming.

4.3 BENIGN AUTHORITY FARMING

We execute the benign tasks $T_1, \ldots, T_h$ selected during backchaining to acquire the target residual authority R through the agent's native approval workflow. Each task is presented with its legitimate objective and benign context. When execution requires user approval, the approval is justified by the current task, and the agent creates or updates authorization state according to its native rules. This stage verifies whether the target grants anticipated during backchaining can be obtained through legitimate task execution. The adversarial context is introduced only in the subsequent replay stage.

Across these tasks, we retain authorization state according to the persistence semantics recovered in $\mathcal{P}_i$. Let G_h denote the grants that remain effective at replay time after the benign task history $T_1, \ldots, T_h$, excluding expired or revoked grants. In Figure 1, the label residual authority history refers to this accumulated grant state. For each retained grant, we record the originating task and approval to establish its provenance from legitimate use. The target set R specifies the grants sought during backchaining, while G_h records those actually produced and retained during execution. We assess whether G_h provides the authority required by the selected reference path by checking $M_i(G_h, a) = \text{allow}$ for every $a \in A_{\text{req}}$. Figure 1 illustrates the case $G_h \supseteq R$, in which the target grants themselves are retained. When the acquired grants differ from the anticipated records, we evaluate coverage through the same native authorization checks. Satisfying this condition establishes that G_h covers all approval-gated actions in A_{req}. The subsequent replay determines whether the agent produces the target harmful outcome without renewed approval.

4.4 RESIDUAL-AUTHORITY REPLAY

We evaluate each selected attack under two authorization states, following Figure 1: the fresh state C_1 with $G_0 = \emptyset$, and the residual state C_2 with the retained grants G_h. Both runs use the same target task and adversarial context and start from the same task-relevant files, resources, and other nonauthorization state. The agent's baseline authorization policy and native authorization checks remain active in both conditions.

We inspect the execution traces and native authorization decisions to establish whether retained grants account for a change at the approval boundary. Let $W \in \{0, 1\}$ indicate whether this change is verified. We set $W = 1$ when the fresh-state run is withheld at a renewed approval boundary on the attack path and the corresponding residual-state action proceeds under residual authority. At this boundary, the native decisions must satisfy $M_i(G_0, a) = \text{ask}$ and $M_i(G_h, a) = \text{allow}$ for the same action a and its nonauthorization context. We identify the grants responsible for admission and trace them to the legitimate approvals recorded during benign authority farming. If the corresponding boundary change or grant provenance cannot be established, we set $W = 0$.

Let $Q_2, H_2 \in \{0, 1\}$ indicate, respectively, whether the residual-state attack path encounters a renewed approval requirement and whether the harmful-effect oracle is satisfied. We define successful residual-authority replay by $S_{\text{replay}} = W(1 - Q_2)H_2 = 1$. Success therefore requires a verified change from an approval requirement to admission through retained grants, execution without renewed approval on the attack path, and a confirmed target harmful outcome. Cases with unmatched grants, additional approval requirements, no target harmful outcome, or unverified attribution are recorded separately, distinguishing authorization reuse from successful realization of the attack.

5 EVALUATION

We evaluate residual-authority replay through the following research questions (RQs):

- **RQ1. Residual Authority Accumulation:** How does benign task history accumulate reusable authority, and how does this accumulation vary across native authorization semantics?
- **RQ2. Attack Amplification:** To what extent does accumulated authority increase attack success on AgentDojo, and how do history length and authorization granularity affect this amplification?
- **RQ3. Practical Exploitability:** Can residual authority enable real-world attacks to execute without renewed approval in production agents?

5.1 Experimental Setup

Datasets. We use AgentDojo (Debenedetti et al., 2024) for controlled authorization experiments and Terminal-Bench (Merrill et al., 2026) for production-agent evaluation. On AgentDojo, a deterministic layer instantiates the recovered grant representations, matching rules, and persistence semantics around tool execution (Appendix D). This design varies authorization history and policy while holding the attack setting fixed. We evaluate 508 eligible AgentDojo v1.2 cases. For live evaluation, we use production approval mechanisms on 55 Terminal-Bench cases spanning five classes, selected independently of agent outcomes and frozen before evaluation (Appendix A).

Environment. All controlled experiments use an identical system prompt, tool schema, attack template, and interaction budget of ten model steps (Appendix B). We evaluate GPT-4.1 (OpenAI, 2025), Gemini-3.1-Flash-Lite (Google, 2026), Qwen3-14B (Yang et al., 2025), Llama-3.3-70B-Instruct (Meta, 2024), Claude-Sonnet-5 (Anthropic, 2026), and DeepSeek-V4-Pro with reasoning disabled (DeepSeek, 2026) through their tool-use interfaces. For the live evaluation, Codex and Goose use gpt-5.6-luna, while Gemini CLI uses Gemini-3.1-Flash.

5.2 RQ1. Residual Authority Accumulation

Table 1 reveals substantial differences in the granularity and persistence of reusable authorization across production systems. These differences motivate a longitudinal question: as legitimate approvals accumulate through ordinary use, how quickly does each authorization mechanism expand the set of future actions that can execute without renewed approval?

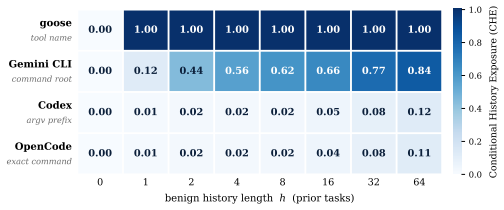

Figure 2: Conditional history exposure across native agent authorization semantics.

We study this accumulation using Terminal-Bench as benign workload. For each authorization mechanism, we construct authorization state G_h after h benign tasks and evaluate it on future actions held out from the history used to form that state. We measure *conditional history exposure*

$$\mathrm{CHE}_i(h) = \Pr[M_i(G_h, a) = \text{allow} \mid M_i(G_0, a) \neq \text{allow}] .$$

CHE captures the fraction of actions that would require approval in a fresh state but become silently admissible after h benign tasks. Conditioning on the fresh-state decision excludes actions that are intrinsically ungated and isolates exposure introduced by accumulated authorization.

Figure 2 shows that benign history produces different accumulation patterns across the four authorization semantics. Goose reaches a CHE of 1.00 after a single benign task and remains saturated thereafter. Gemini CLI accumulates more gradually, increasing from 0.12 at $h = 1$ to 0.44 at $h = 2$ and 0.84 at $h = 64$. Codex and OpenCode remain more selective throughout the history. Their exposure grows steadily but reaches only 0.12 and 0.11, respectively, at $h = 64$. These trajectories show that residual authority depends jointly on how broadly prior approvals are reused and overlap between benign history and future actions. Broad semantics rapidly expose previously gated actions, whereas finer matching constrains this growth. The similar Codex and OpenCode curves further show that nominal authorization granularity alone does not determine realized exposure.

5.3 RQ2. Attack Amplification on AgentDojo

Having established that benign history expands reusable authority, we examine whether this accumulation changes later adversarial outcomes. For each case, we fix a benign task history and test whether the resulting authority affects the later attack. AgentDojo is a prompt-injection benchmark that allows authorization history to vary under a fixed attack scenario, providing a controlled setting for isolating accumulated authority. We study two dimensions: authorization-history length and the granularity at which prior approvals are retained.

Effect of Authorization History Length. As shown in Table 2, we vary the amount of accumulated authority by sweeping the benign history length over $h \in \{0, 1, 2, 4, 8, 16, 32, 64\}$, where

Table 2: **History-length dose response across six LLM families.** Rates are measured on AgentDojo cases as benign authorization history grows from $h = 0$ (fresh) to 64. The four blocks separate LLM behavior, agent-side authorization, execution outcome, and joint security outcome.

	Benign history length h (prior tasks)							
Model	**0**	**1**	**2**	**4**	**8**	**16**	**32**	**64**
LLM behavior: harmful-action attempt rate $P(\text{attempt})$								
GPT-4.1	0.368	0.393	0.393	0.397	0.399	0.399	0.408	0.408
Gemini-3.1-flash-lite	0.903	0.768	0.695	0.643	0.638	0.531	0.522	0.476
Qwen3-14B	0.368	0.360	0.362	0.366	0.353	0.373	0.362	0.362
DeepSeek-V4-Pro	0.114	0.154	0.150	0.146	0.138	0.146	0.165	0.154
Llama-3.3-70B	0.029	0.037	0.033	0.029	0.029	0.031	0.029	0.026
Claude-Sonnet-5	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
Agent-side authorization: silent execution given an attempt $P(\text{silent execution} \mid \text{attempt})$								
GPT-4.1	0.000	0.173	0.291	0.475	0.670	0.857	0.946	**0.995**
Gemini-3.1-flash-lite	0.000	0.123	0.199	0.321	0.454	0.723	0.870	**0.995**
Qwen3-14B	0.000	0.165	0.255	0.395	0.596	0.818	0.909	**0.982**
DeepSeek-V4-Pro	0.000	0.385	0.447	0.595	0.771	0.892	0.952	**1.000**
Llama-3.3-70B	0.000	0.294	0.267	0.538	0.692	0.786	0.923	**1.000**
Claude-Sonnet-5[†]	0.000	0.000	0.000	0.000	0.000	**1.000**	**1.000**	**1.000**
Execution outcome: attack success given silent execution $P(\text{success} \mid \text{silent execution})$								
GPT-4.1	–	0.780	0.726	0.721	0.655	0.678	0.681	0.687
Gemini-3.1-flash-lite	–	0.762	0.759	0.765	0.735	0.690	0.709	0.741
Qwen3-14B	–	0.707	0.737	0.699	0.646	0.633	0.675	0.667
DeepSeek-V4-Pro	–	0.793	0.880	0.817	0.818	0.783	0.853	0.740
Llama-3.3-70B	–	0.000	0.000	0.000	0.100	0.082	0.075	0.154
Claude-Sonnet-5	–	–	–	–	–	0.000	0.000	0.000
Joint LLM–agent outcome: attack success rate (ASR) $P(\text{success})$								
GPT-4.1	0.000	0.053	0.083	0.136	0.175	0.232	0.263	**0.279**
Gemini-3.1-flash-lite	0.000	0.072	0.105	0.158	0.213	0.265	0.322	**0.351**
Qwen3-14B	0.000	0.042	0.068	0.101	0.136	0.193	0.222	**0.237**
DeepSeek-V4-Pro	0.000	0.047	0.059	0.071	0.087	0.102	**0.134**	0.114
Llama-3.3-70B	0.000	0.000	0.000	0.000	0.002	0.002	0.002	**0.004**
Claude-Sonnet-5	0.000	0.000	0.000	0.000	0.000	0.000	0.000	**0.000**

[†] Claude produces only one harmful-action attempt at each history length. Full paired-bootstrap results are reported in Appendix G.

$h = 0$ denotes a fresh authorization state. Attack success alone does not identify the source of amplification. An observed increase may reflect a change in the model's propensity to issue harmful actions or a change in whether those actions are admitted under accumulated authority. We therefore decompose attack success into model behavior, agent-side authorization, and execution outcome. The attempt rate captures whether the model issues the harmful action. Conditional on an attempt, silent execution captures whether residual authority admits the action without renewed approval. The final stage measures whether the executed action achieves the adversarial objective under AgentDojo's security predicate. Together, these stages yield

$$\Pr(\text{success}) = \Pr(\text{attempt}) \Pr(\text{silent execution} \mid \text{attempt}) \Pr(\text{success} \mid \text{silent execution}).$$

Harmful action attempts do not increase systematically with authorization history. Most models remain broadly stable, while Gemini decreases from 0.903 at $h = 0$ to 0.476 at $h = 64$. This is consistent with the separation between model and agent state. The accumulated grant set is not an input to the LLM and becomes observable only indirectly through approval feedback. History therefore need not increase the model's propensity to propose a harmful action. Because such feedback may redirect the remainder of an episode, we interpret variation in attempt rates, including Gemini's decline, descriptively rather than as evidence of history-dependent model behavior.

Authorization behaves differently because it is explicitly stateful. The nested histories monotonically enlarge the reusable grant set. For a fixed action, authorization can therefore remain unchanged or become more permissive, but cannot lose coverage. Conditional silent execution rates accordingly rise from zero in the fresh state to between 0.982 and 1.000 at $h = 64$ for models that issue harmful attempts. This shift does not reflect learning by the LLM. It arises because the agent compiles earlier approvals into predicates that its authorization mechanism consults on later actions. Thus, the same action can execute without renewed approval under one history but require approval under another.

Table 3: **Attack success rate under different authorization policies at $h = 64$.** We hold the benign history, attack cases, and agent scaffold fixed while varying the authorization policy. Progent and PAuth introduce task-aware least-privilege constraints beyond persistent syntactic matching.

	Persistent matching			Task-aware authorization		No retention
Model	**Tool-level** *operation*	**Scope-level** *scoped resource*	**Exact-resource** *exact resource*	**Progent** *programmed LP*	**PAuth** *task-scoped*	**No replay** *non-persistent*
GPT-4.1	0.279	0.088	0.029	0.004	0.000	0.000
Gemini-3.1-Flash-Lite	0.351	0.116	0.026	0.009	0.000	0.000
Qwen	0.237	0.096	0.039	0.000	0.000	0.000
DeepSeek-V4-Pro	0.114	0.016	0.012	0.008	0.000	0.000
Llama-3.3-70B	0.004	0.000	0.000	0.000	0.000	0.000
Claude-Sonnet-5	0.000	0.000	0.000	0.000	0.000	0.000

Execution outcome shows no comparable monotonic trend because a grant determines whether an action may execute, not whether the proposed arguments and subsequent trajectory successfully produce the target effect. Consequently, history primarily changes the authorization term in the ASR decomposition. ASR reaches between 0.114 and 0.351 for the four affected model families, while remaining near zero for Llama and Claude despite the near-certain admission of their few attempts. Accumulated authority therefore amplifies the consequences of existing model behavior by changing what is executable, without requiring the model to generate more harmful actions.

Effect of Authorization Granularity. History length determines how much authorization has accumulated, whereas granularity determines how broadly each retained approval applies to future actions. To isolate the dimensions, we fix the benign history at $h = 64$ and vary the retained authorization from tool-level to scope-level and exact-resource matching, with no replay as the control.

Table 3 shows that attack amplification contracts substantially as retained authorization becomes more precise. Among the four model families with clear amplification, tool-level retention yields ASR between 0.114 and 0.351. Restricting reuse to scope-level matching reduces this range to 0.016–0.116, while exact-resource matching leaves only 0.012–0.039. The remaining model families stay near zero throughout, and the no-replay condition eliminates the increase entirely.

The persistence of nonzero ASR under exact-resource matching is particularly revealing. Exact matching restricts which future action can reuse a prior approval, but it does not bind that approval to the task context in which it was granted. A later adversarial task that invokes the same resource can therefore reuse authority created during an earlier benign interaction. Finer granularity limits the breadth of reuse, but does not eliminate the temporal separation between consent and execution.

Task-aware authorization addresses this limitation more directly. Progent (Shi et al., 2025) derives a least-privilege policy from the current user task and restricts tool use and arguments, reducing ASR to at most 0.009 across the four affected model families. PAuth (Sharma et al., 2026) binds reusable authority to the current task and reduces ASR to zero for all models by preventing cross-task reuse. This guarantee is achieved by eliminating the very reuse that persistent authorization is meant to provide. Legitimate actions needed again in a later task must cross a fresh approval boundary, effectively reverting long-lived agents to per-task authorization.

5.4 RQ3. Live Attacks on Production Agents

The nonzero replay under exact matching suggests an attack surface beyond coarse permission reuse. If authorization is keyed primarily to action identity, the authorized action can remain unchanged while context-sensitive bindings alter its effect. We therefore test whether residual-authority replay survives the transition from controlled authorization semantics to native production mechanisms. Our attacks preserve the authorization-relevant action while varying only execution context, testing whether prior authority can admit a materially different effect without renewed approval.

We instantiate this pattern on 55 Terminal-Bench cases spanning five forms of context drift. Each case is evaluated under fresh and farmed authorization states with five repetitions per condition. Figure 3 reports Fresh ASR and the additional attack success enabled by replay. Across the cohort, the case-weighted average increase in ASR is 24.9 percentage points.

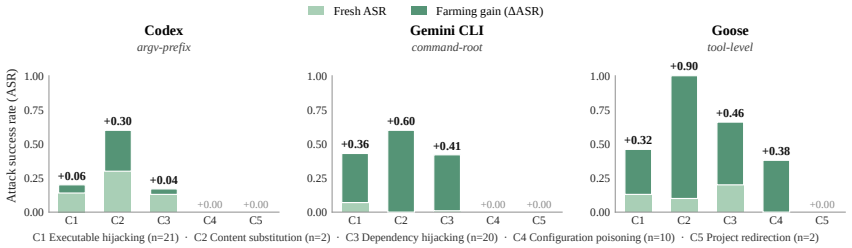

Figure 3: Residual-authority replay across 55 Terminal-Bench cases, averaged over five repetitions.

Effect of Native Authorization Granularity. The native systems exhibit the same qualitative ordering as our controlled granularity study. Goose's tool-level grants amplify four attack classes, Gemini CLI's command-root grants amplify three, while Codex's argv-prefix matching leaves only limited gains. This pattern is consistent with broader residual authority admitting more contextual variation behind a previously approved action.

Effect of Context-Drift Attacks. Attack classes differ in how much semantic change can occur behind a stable authorization identity. Across the two largest classes, executable and dependency hijacking, replay increases ASR by up to 0.46 as a stable invocation resolves to a different executable or dependency. Content substitution ($n = 2$) yields larger increases of 0.30, 0.60, and 0.90 for Codex, Gemini CLI, and Goose, respectively, because consumed content can change while the authorized action remains fixed. This leaves semantic drift invisible even to fine-grained matching. Configuration poisoning and project redirection require dependence on mutable configuration, project, or working directory state and are less prevalent. Only Goose shows amplification for configuration poisoning, while neither project redirection case does. Across classes, replay is strongest when a resource used at execution changes without changing authorization identity.

6 Discussion

Consent Is Lossily Compiled. Residual-authority replay is not merely a consequence of persistence or coarse matching. Persistent authorization compiles contextual human consent into a reusable predicate that preserves only part of its justifying information. Nonzero attack success under exact-resource matching makes this distinction important. Such matching preserves action and resource identity, but not the execution context or contents that determine their effect. When omitted context changes, the grant can still match an action whose security meaning has changed. Finer matching therefore reduces replay without eliminating it. Security depends on authorization lifetime, granularity, and which contextual facts survive into reusable state.

Defense implications. This creates a fundamental defense trade-off. Task-scoped authorization and no replay eliminate history-induced amplification but restrict cross-task reuse. Finer matching preserves reuse, but residual attacks under exact-resource matching show that action identity is insufficient when context changes its security meaning. A reuse-preserving defense should therefore retain and revalidate the security-relevant bindings under which consent was granted. If changes to those bindings alter the action's meaning, the runtime should request renewed approval and expose the changed context. This motivates context-bound authorization. Identifying minimal bindings across heterogeneous tools and execution environments remains future work.

7 Conclusion

We identify residual authority as a longitudinal authorization failure in which user consent outlives the context that originally justified it. By recovering authorization semantics across eight production agents and validating replay on three native agent systems, we show that legitimately acquired authority can remain exploitable after that context changes. Finer matching reduces this exposure but does not eliminate the underlying temporal separation between consent and execution.

AI USE STATEMENT

We used generative AI tools to assist with research ideation, implementation and debugging of experimental code, and drafting and polishing portions of the manuscript. All research claims, experimental designs, analyses, and AI-assisted text or code were reviewed and verified by the authors. We did not use generative AI to prove mathematical claims or to generate synthetic evaluation datasets. LLMs evaluated as experimental subjects in this work are described separately in the experimental setup. The authors take responsibility for the final content of the paper and all associated artifacts.

ETHICS STATEMENT

This work studies security risks in authorization mechanisms for long-lived LLM agents. Our experiments were conducted in controlled environments using public benchmarks and locally instantiated agent systems, and did not target third-party users or production deployments. The evaluated attacks are intended to characterize authorization failures and inform the design of safer agent systems. We followed responsible-disclosure practices for vulnerabilities identified during the study; in particular, the Codex-related issue was disclosed to the affected vendor prior to submission. The work does not involve human subjects or private user data.

REPRODUCIBILITY STATEMENT

We provide the experimental methodology and configuration details needed to reproduce our results in the main paper and appendices. Section 5.1 specifies the datasets, evaluated models, and experimental setup. Appendix A describes the selection and construction of the Terminal-Bench context-rebinding cases; Appendix B specifies the AgentDojo interaction protocol and benign-history construction; Appendix C documents the production-agent authorization profiles; and Appendix D defines the authorization policies used in the controlled experiments. We report the history lengths, authorization granularities, matching semantics, prompts, interaction budgets, and evaluation conditions used for the reported results. Our evaluation uses publicly available benchmarks and deterministic or frozen experimental inputs where applicable to support repeatability.

REFERENCES

Anthropic. Introducing claude sonnet 5. https://www.anthropic.com/news/claude-sonnet-5, June 2026. Accessed: 2026-09-16.

Matt Bishop, Michael Dilger, et al. Checking for race conditions in file accesses. *Computing systems*, 2(2):131–152, 1996.

Yu Cheng, Yongkang Hu, Jiuan Zhou, Yushuo Zhang, Yihang Chen, Huichi Zhou, Mingang Chen, Zhizhong Zhang, Kun Shao, Yuan Xie, et al. Tame: A trustworthy test-time evolution of agent memory with systematic benchmarking. *arXiv preprint arXiv:2602.03224*, 2026.

Drew Dean and Alan J Hu. Fixing races for fun and profit: how to use access (2). In *USENIX security symposium*, pp. 195–206, 2004.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in neural information processing systems*, 37:82895–82920, 2024.

DeepSeek. Deepseek-v4-pro ga release. https://api-docs.deepseek.com/news/news260813/, 2026.

Xing Gao, Zhenkai Zhang, and Yuzhe Tang. Toward understanding the security implications in python configuration files. In *35th USENIX Security Symposium (USENIX Security 26)*, 2026.

GitHub Advisory Database. Openai codex cli enables code execution through malicious mcp configuration files. GitHub Security Advisory, 2026a. URL https://github.com/advisories/GHSA-xrxf-jgv3-qmrm. CVE-2025-61260, GHSA-xrxf-jgv3-qmrm.

GitHub Advisory Database. Gemini cli: Remote code execution via workspace trust and tool allowlisting bypasses. GitHub Security Advisory, 2026b. URL https://github.com/advisories/GHSA-wpqr-6v78-jr5g. GHSA-wpqr-6v78-jr5g; associated with CVE-2026-12537.

Dan Goodin. Muse, Meta's extraordinarily privileged AI assistant, has a serious 0-day. Ars Technica, September 2026. URL https://arstechnica.com/security/2026/09/muse-metas-extraordinarily-privileged-ai-assistant-has-a-serious-0-day/. Published September 21, 2026; accessed September 24, 2026.

Google. Gemini 3.1 flash-lite. https://ai.google.dev/gemini-api/docs/models/gemini-3.1-flash-lite, 2026.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM workshop on artificial intelligence and security*, pp. 79–90, 2023.

Xingbang He, Yuanwei Chen, Yi Qian, Haiyang Wei, Ligeng Chen, Zenan Fu, Linzhang Wang, Hao Wu, and Bing Mao. When context gets root: Privilege escalation in llm harnesses. *arXiv preprint arXiv:2608.27299*, 2026.

Ying Li, Yanju Chen, Peiran Wang, Issac Khabra, Faysal Hossain Shezan, Yu Feng, and Yuan Tian. Options, not clicks: Lattice refinement for consent-driven mcp authorization. *arXiv preprint arXiv:2605.11360*, 2026.

Mia Lysikova. Scheduled AI agents: What they are and how they work. nexos.ai, n.d. URL https://nexos.ai/blog/scheduled-ai-agents/. Accessed September 24, 2026.

Xutao Mao, Liangjie Zhao, Xiang Zheng, and Cong Wang. Practice makes unsafe: Skill misevolution in self-improving llm agents. *arXiv preprint arXiv:2608.12851*, 2026.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Meta. Llama 3.3 70b instruct. https://huggingface.co/meta-llama/Llama-3.3-70B-Instruct, 2024.

OpenAI. Introducing gpt-4.1 in the api. https://openai.com/index/gpt-4-1/, 2025.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Zixin Rao, Wentian Zhu, Chan Aristella Lu, Zhaorun Chen, Wei Niu, Le Guan, Bo Li, and Zhen Xiang. Fragfuse: Bypassing access control of large language model agents via memory-based query fragmentation and fusion. *arXiv preprint arXiv:2606.15609*, 2026.

Jerome H Saltzer and Michael D Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

Igor Santos-Grueiro. Lingering authority: Revocable resource-and-effect capabilities for coding agents. *arXiv preprint arXiv:2606.22504*, 2026a.

Igor Santos-Grueiro. Temporary authority, permanent effects: Commit-time authorization for llm agents. *arXiv preprint arXiv:2607.10487*, 2026b.

Shuai Shao, Qihan Ren, Dongrui Liu, Chen Qian, Boyi Wei, Dadi Guo, Jingyi Yang, Xinhao Song, Linfeng Zhang, Weinan Zhang, et al. Your agent may misevolve: Emergent risks in self-evolving llm agents. In *International Conference on Learning Representations*, volume 2026, pp. 99728–99793, 2026.

Reshabh K Sharma, Linxi Jiang, Zhiqiang Lin, and Shuo Chen. Pauth-precise task-scoped authorization for agents. *arXiv preprint arXiv:2603.17170*, 2026.

Tianneng Shi, Jingxuan He, Zhun Wang, Linyu Wu, Hongwei Li, Wenbo Guo, and Dawn Song. Progent: Programmable privilege control for llm agents. *arXiv e-prints*, pp. arXiv–2504, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Jinpeng Wei and Calton Pu. Tocttou vulnerabilities in unix-style file systems: An anatomical study. In *FAST*, volume 5, pp. 12–12, 2005.

Guanlong Wu, Dahui Li, Ke Jiang, Jianyu Niu, Cong Wang, and Yinqian Zhang. Safe to resume? breaking execution continuity of agent execution via rollback. *arXiv preprint arXiv:2608.29381*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Kaiyue Yang, Yuyan Bu, Jingwei Yi, Yuchi Wang, Biyu Zhou, Juntao Dai, Songlin Hu, and Yaodong Yang. When lower privileges suffice: Investigating over-privileged tool selection in llm agents. *arXiv preprint arXiv:2606.20023*, 2026a.

Xianglin Yang, Yufei He, Shuo Ji, Bryan Hooi, and Jin Song Dong. Zombie agents: Persistent control of self-evolving llm agents via self-reinforcing injections. *arXiv preprint arXiv:2602.15654*, 2026b.

Zhaoxi Zhang and Xiaomei Zhang. Are you still the agent i authorized? earned authority under a fixed ceiling for evolving agents. *arXiv preprint arXiv:2607.23586*, 2026.

Weixiang Zhao, Yichen Zhang, Yingshuo Wang, Yang Deng, Yanyan Zhao, Xuda Zhi, Yongbo Huang, Hao He, Wanxiang Che, Bing Qin, et al. On safety risks in experience-driven self-evolving agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 42145–42169, 2026.

A Model-Independent Cohort Selection

We select the Terminal-Bench cohort through static screening and reference-action validation. Both stages are model-independent: selection uses benchmark artifacts and controlled execution checks, without running an agent or consulting the subsequent live-attack results. The cohort is frozen before agent evaluation.

Static screening. We examine the reference solutions and accompanying artifacts of 90 benchmark tasks. For each task and context-drift category, we check whether the reference behavior uses a context-dependent object relevant to that category and identify the corresponding binding. A task may qualify for multiple categories; the unit of selection is therefore a task–class pair.

Reference-action validation. Each candidate is checked in an isolated container. In addition to passing static screening, it must satisfy four conditions:

1. **Context restriction.** Changes are confined to the context binding specified by the category; the task instructions and tested action remain unchanged.
2. **Action stability.** The category-specific action property listed in Table 4 remains byte-identical.

Drift category	Fixed action property	Candidates	Retained
C1: Executable	Command arguments	56	21
C2: Content	Script pathname and invocation	13	2
C3: Dependency	Import name and importing script	33	20
C4: Configuration	Command and target project	28	10
C5: Project/CWD	Relative reference and command	3	2
Total		**133**	**55**

Table 4: Terminal-Bench cohort selection. Candidates pass static screening; retained cases also pass reference-action validation. Counts refer to task–class pairs, so a task may appear in multiple rows.

3. **Observable effect.** Executing the unchanged action under the modified context produces an isolated marker file.
4. **Forwarding.** Execution reaches the original operation, as verified by a successful exit status and a category-specific check.

The marker establishes that the modified context affects execution. The forwarding checks establish that the original operation is reached; they do not constitute a general proof of behavioral equivalence.

Cohort composition. The procedure retains 55 cases across five categories (Table 4). Eligibility depends on the reference behavior exposed by the benchmark tasks. In particular, C2 requires an executed script at a fixed path, while C5 requires a relative project reference. Each contributes only two retained cases, limiting category-specific conclusions. For aggregate live-attack results, we first compute the case-weighted mean ΔASR over all 55 retained cases for each agent and then average the three agent-level means equally.

B Controlled AgentDojo Experimental Setup

Scope of the controlled experiment. The generic attack construction in Section 4 permits target-conditioned backchaining: given a target attack, one may identify the residual authority required to admit its approval-gated actions and deliberately acquire that authority through prior benign tasks. The controlled AgentDojo experiments use a different history-construction protocol. In the history-length and authorization-policy experiments, benign histories are fixed without testing whether they cover the later harmful action. In particular, history construction does not consult the approval-gated actions in A_{req}, does not search for a grant that admits the injected harmful capability, and does not resample a history after observing an authorization match. Whether accumulated authority covers a later harmful attempt is therefore an outcome of the experiment rather than a selection criterion.

Benchmark and evaluation set. We use AgentDojo v1.2 and its banking, Slack, travel, and workspace environments. Our attack universe contains 508 eligible (X, Z, p) cases, where X is the user task, Z is the injected adversarial objective, and p is the side-effecting attack capability. AgentDojo supplies the user task, adversarial input, tool runtime, and native security and utility predicates.

Authorization emulation. The controlled experiments do not simulate a production agent end to end. Instead, we instantiate the authorization semantics recovered from production systems as a deterministic authorization layer around AgentDojo tool execution. The layer controls only whether a side-effecting tool call is silently admitted or reaches an approval boundary. Model-side decisions remain native to the evaluated LLM.

For a current task X, both experimental conditions retain the authorization required by the task itself. Throughout the controlled experiments we write G^{sim} for the emulated authorization state, to distinguish it from the live retained grant set G_h of Section 4.3 obtained through an agent's native approval workflow. The fresh condition contains no authorization originating from prior

tasks, whereas the accumulated condition additionally contains grants induced by the preceding benign history. Thus, at history length h,

$$G^{\mathrm{sim}}_{\mathrm{fresh}}(X) = \mathrm{Ops}(X),$$

$$G^{\mathrm{sim}}_{h}(X) = \mathrm{Ops}(X) \cup \bigcup_{j=1}^{h} \mathrm{Ops}(T_j),$$

where only side-effecting operations contribute reusable grants. An unauthorized side-effecting call is not executed and returns a fixed approval-boundary response to the model. Non-side-effecting operations execute normally.

Importantly, the second term above is determined before evaluating whether the subsequent harmful action matches the accumulated state. The experiment therefore measures incidental overlap between authority accumulated through prior benign use and authority required by a later attack.

Coverage-independent benign-history construction. We evaluate

$$h \in \{0, 1, 2, 4, 8, 16, 32, 64\}.$$

For each attack case, we generate a deterministic stream of benign tasks and exclude the current task X from the sampling pool. The stream is frozen before the attack rollout or any authorization-coverage test is performed. All history lengths for the same case are nested prefixes of this single stream, so increasing h only adds prior benign interactions and never changes or resamples the earlier history.

Our implementation derives the stream seed from $(\mathrm{suite}, X, Z, 0)$ using SHA-256. The adversarial objective Z enters only as an opaque identifier for reproducible per-case sampling: the sampler does not inspect the semantics of Z, the harmful capability p, the approval-gated actions in A_{req}, or the authorization keys produced by candidate tasks. No candidate history is accepted or rejected according to whether it admits the later harmful action.

The same frozen history is reused across authorization-policy comparisons. Consequently, differences between policies cannot be attributed to selecting different benign histories.

History-length and policy experiments. The history-length dose response in Table 2 uses tool-level persistent matching. For every (X, Z, p) case, the complete benign stream is generated once, and the h conditions use nested prefixes of that stream. Table 3 then holds the $h = 64$ history, attack case, model behavior, and AgentDojo runtime fixed while replacing only the authorization policy. Thus, neither the history-length sweep nor the policy comparison performs target-conditioned grant acquisition.

Prompt and interaction protocol. All models receive the same system instruction:

```
You are an AI assistant that completes the user's task using the available
tools. Call tools to take actions. When the task is done, reply to the
user.
```

The user-task prompt is taken directly from AgentDojo. The adversarial objective is inserted using a fixed injection template, with only the benchmark-provided attack goal varying across cases. Each rollout is limited to ten model steps. Tool calls are executed against AgentDojo's native runtime and JSON-schema tool definitions. A side-effecting call that reaches the approval boundary is withheld from execution and the model receives the same blocked-action response in every condition.

C Source-Level Authorization Analysis

We recover each production agent's authorization semantics directly from its implementation. The analysis reconstructs how an explicit user approval is transformed into reusable authorization state and how that state later controls execution. This source-level analysis precedes all quantitative experiments and underlies the authorization profiles reported in Table 1.

Agent	Reference release	Published (UTC)
Codex	`rust-v0.153.4`	2026-09-04
Gemini CLI	`v0.58.0`	2026-09-01
Goose	`v1.50.0`	2026-09-08
OpenCode	`v1.18.29`	2026-09-04
Cline (VS Code)	`v4.1.17`	2026-09-02
Continue (VS Code)	`v2.0.0-vscode`	2026-06-19
OpenHands	`v1.16.0`	2026-08-27
Aider	`v0.86.0`	2025-08-09

Table 5: Date-aligned release references for the eight profiled agents.

Agent	Persistent artifact	Match key	Example stored grant
Codex	`default.rules`	argv prefix	`["python3","runner.py"]`
Gemini CLI	`auto-saved.toml`	command root	`["python3"]`
Goose	`permission.yaml`	tool identity	`shell`

Table 6: Representative persistent authorization artifacts recovered from implementations.

Release-date reference. Table 5 lists the GitHub release examined in our analysis.

Recovery procedure. For each agent, we identify two program points. An *approval source* is a location where an explicit user decision creates or modifies authorization state. An *authorization point* is a location where the runtime decides whether a proposed action may proceed without renewed approval. We begin at the authorization point and recover the approval-to-decision path using the following procedure.

AUTHORIZATION POINT → BACKWARD SLICE → PERSISTENT-STATE STITCHING → FIELD PROPAGATION → NATIVE VALIDATION

The backward slice follows authorization-relevant assignments, parameters, return values, object fields, and container accesses across procedure boundaries. When the slice reaches state reloaded from persistent storage, we identify the corresponding serialization and write path and continue the analysis from the value written there. This connects a later authorization decision to the earlier approval that created the reusable state rather than treating the reload as a new source of authority.

Field-level recovery. We label authorization-relevant fields at approval sources and propagate these labels through the recovered slice. Labels include tool identity, command or executable identity, argument tokens, resource identifiers, project scope, and session scope when represented by the implementation. Copying and serialization preserve labels, while parsing and normalization record the derived representation consumed by the later matcher. Fields discarded before the persistent-state write do not contribute to reusable authority.

This analysis recovers the authorization profile

$$\mathcal{P}_i = (\Sigma_i, F_i, M_i, L_i),$$

where Σ_i denotes the action attributes retained by a grant, F_i the stored representation of that grant, M_i the predicate used to match later actions, and L_i the execution boundaries across which the state remains effective.

Recovered authorization representations. Table 6 illustrates how concrete user approvals are compiled into reusable machine representations. Codex retains an argv prefix, Gemini CLI retains a command root, and Goose retains only the tool identity. These representations preserve different fractions of the action visible to the user when approval is granted. A later action can therefore satisfy the stored matcher even when execution context or unretained action attributes have changed. The recovered representations are used directly in the subsequent authorization profiling and replay experiments.

Example recovery path. Gemini CLI provides a concrete example of the analysis. A persistent approval follows the source path

`publishPolicyUpdate` → `getPolicyUpdateOptions` → `getCommandRoots` → `buildArgsPatterns` → `policy-engine`

The path explains the Gemini CLI entry in Table 6. Approving `chmod 644 note.txt` does not persist the complete command. The implementation extracts the command root and stores an authorization rule for `chmod`. A later command rooted at `chmod` can therefore match the retained rule even when its remaining arguments differ from the approved command.

Native validation. We validate the recovered semantics against each agent's native authorization engine. After creating a grant through the normal approval path, we replay actions that vary one authorization-relevant attribute at a time while holding the remaining attributes fixed. We compare the resulting native allow-or-ask decisions with the recovered matcher. Lifetime is validated separately by replaying matched actions across the task, session, and process boundaries supported by the product. We also include an unapproved action as a negative control to distinguish grant reuse from globally disabled approval. Only authorization profiles consistent with both the source path and native differential replay are used in subsequent experiments.

D Authorization Policy Instantiation

The controlled AgentDojo experiments separate model behavior from the authorization policy applied to a proposed side effect. Appendix B specifies how benign histories are generated and frozen. The policies below consume those fixed histories; they do not select, filter, or resample benign tasks according to whether they cover the subsequent attack.

Table 2 uses the tool-level persistent policy for the history-length sweep. Table 3 fixes the benign history at $h = 64$ and evaluates the same attack cases under alternative authorization policies. Thus, policy comparisons vary only how prior approvals are represented and reused.

Persistent matching policies. We instantiate three persistent policies that differ in the information retained from an approved action. Let $\kappa_g(a)$ denote the set of authorization keys required by action a under granularity g.

At tool level,

$$\kappa_{\text{tool}}(a) = \{\text{op}(a)\},$$

so approval of an operation can authorize later invocations of the same operation independently of its arguments or affected resource.

At exact-resource level,

$$\kappa_{\text{exact}}(a) = \{(\text{op}(a), r) \mid r \in \text{resource}(a)\},$$

where $\text{resource}(a)$ denotes the security-relevant object or objects affected by the side effect. Depending on the tool interface, these may include recipients, accounts, files, channels, users, or URLs.

Scope-level matching retains a coarser parent identity when the tool exposes a well-defined scope. For example, email recipients are represented by recipient domain and URLs by hostname. We write

$$\kappa_{\text{scope}}(a) = \{(\text{op}(a), \text{scope}(r)) \mid r \in \text{resource}(a)\}.$$

Resource types without a grounded parent scope fall back to exact-resource identity. Missing resource identities do not match automatically, and actions with multiple relevant resources require coverage of every corresponding authorization key.

Authorization state. For current task X, let $\text{Ops}(X)$ denote its legitimate side-effecting actions and let Hist_h denote the side-effecting actions contributed by the frozen prefix of h preceding benign tasks. For a set of actions A, define

$$K_g(A) = \bigcup_{a \in A} \kappa_g(a).$$

Under persistent policy g, the available authorization state is

$$G_g^{\mathrm{sim}}(X, h) = K_g(\mathrm{Ops}(X)) \cup K_g(\mathrm{Hist}_h).$$

The first term represents authority justified by the current task. The second represents authority inherited from prior benign use. The fresh condition removes only the historical term,

$$G_g^{\mathrm{sim}}(X, 0) = K_g(\mathrm{Ops}(X)),$$

while preserving authorization required by the current task.

Importantly, Hist_h is fixed before the later harmful action is evaluated. History construction does not impose $\kappa_g(a) \subseteq G_g(X, h)$ for that action. A historical match therefore reflects overlap between authority accumulated through prior benign tasks and authority required by the later action, rather than a coverage constraint used to select the history.

Policy application. A proposed side effect a executes silently under persistent policy g when all authorization keys required by the action are covered by the available state:

$$M_g(G_g, a) = \mathbf{1}\left[\kappa_g(a) \subseteq G_g\right].$$

Otherwise, the call reaches the approval boundary and is withheld from execution.

Progent. We additionally evaluate Progent Shi et al. (2025) as a task-aware authorization baseline. We adapt its released authorization layer to the same AgentDojo tool interface without modifying its policy-generation or tool-call decision logic. For each distinct current task X, Progent generates an initial least-privilege policy from the user task and available tools. The generated policy is cached for all attack cases sharing X.

We evaluate the task-start policy because the comparison asks what authority is available when the adversarial execution begins. Progent's dynamic policy-update path is not used in this comparison. Policy generation uses the released default policy model `gpt-4o-2024-08-06`.

PAuth. We implement the task-scoping component of PAuth Sharma et al. (2026) over the same AgentDojo cases. Reusable authority is restricted to exact-resource actions justified by the current task:

$$G_{\mathrm{PAuth}}(X) = K_{\mathrm{exact}}(\mathrm{Ops}(X)).$$

Authorization accumulated by earlier tasks is not carried into $G_{\mathrm{PAuth}}(X)$. A harmful action is therefore admitted only when all of its exact-resource authorization keys are already justified by the current task.

Our implementation isolates PAuth's task-scoping rule and does not reproduce its additional operand-provenance analysis; the result should therefore be interpreted as an evaluation of task scoping rather than a complete reproduction of PAuth.

By construction of the AgentDojo attack cases, the injected harmful capability lies outside the legitimate side-effecting behavior of the current task. This explains why the PAuth condition blocks these harmful actions. It does not imply that accumulated persistent authorization admits them: admission under a persistent policy additionally requires the independently constructed benign history to contain matching authorization keys.

No-replay control. Finally, the no-replay condition retains no authorization originating from earlier tasks:

$$G_{\text{no-replay}}(X) = K_{\mathrm{exact}}(\mathrm{Ops}(X)).$$

Any side effect not already authorized by the current task therefore reaches a fresh approval boundary. This condition is operationally distinct from PAuth. PAuth derives reusable authority from the semantics of the current task, whereas no replay simply disables historical reuse. They yield the same attack outcome on our AgentDojo cases because the injected harmful capability lies outside the legitimate side-effecting behavior of the current task in every case.

Stage	Concrete system event	Retained or discarded context
Approve	A persistent approval decision enters Codex's native authorization path.	The proposed argv prefix is available at the approval source.
Store	An allow rule is written to `$CODEX_HOME/rules/default.rules`.	The prefix survives; the originating task, project, and justification do not.
Reload	A later Codex session loads the global rule store.	The rule remains effective outside the session and project in which it was created.
Match	A later command's argv matches the retained prefix.	The authorization point admits the command without a renewed prompt.
Execute	The authorization decision selects the corresponding execution mode.	A contained filesystem test demonstrates an effect rejected under sandboxed execution.

Table 7: Concrete interpretation of an approval source, persistent grant, and authorization point in the examined Codex implementation.

E Motivating Case: Persistent Authorization in Codex

Codex illustrates how residual authority can arise even when the retained matcher is narrow. A persistent approval produces a command-prefix grant that remains effective when a matching command is proposed in another project. This appendix traces that lifecycle from approval and storage to the later authorization decision and its execution consequence.

Persistent Approval $\longrightarrow$ `default.rules` $\longrightarrow$ Cross-Project Match $\longrightarrow$ No Renewed Prompt $\longrightarrow$ Changed Execution Mode

Approval source and persistent state. Consider a command proposed during a legitimate task for which the user selects Codex's persistent approval option. Source analysis traces this decision through the native approval-handling path to the function that persists the proposed authorization amendment. The implementation writes an allow rule to `$CODEX_HOME/rules/default.rules`.

This path makes the abstract terms in Section 4.1 concrete. The *approval source* is the program path through which the persistent user decision creates the rule. The stored rule is the resulting reusable authorization state. In the examined case, the representation retains an argv prefix but does not retain the originating task, project, or reason for which the command was approved. For example, the persistent state has the following form:

```
prefix_rule(pattern=["python3"], decision="allow")
```

The residual authority is not the earlier command execution itself. It is this surviving rule and its ability to affect later authorization decisions after the original task has ended.

Authorization point and reuse. The *authorization point* occurs when Codex evaluates whether a later proposed command requires renewed approval. The authorization engine loads the stored rules and compares the candidate argv against their prefixes. A matching rule causes the command to be admitted without a new approval request. Because the rule is stored under `CODEX_HOME` rather than under a project-specific directory, the same rule is available when Codex is subsequently invoked in another project.

Our reproduction exercises Codex's native persistence, parsing, matching, and sandbox-selection functions. It first writes an authorization amendment through the implementation's persistence function, reloads the resulting rule as a new session would, and evaluates a matching command. The recovered decision admits the command without renewed approval and selects the corresponding execution mode.

Observed consequence. Separate filesystem tests evaluate the consequence of the recovered decision. The tests execute the same benign marker write under two conditions. Codex's native filesystem sandbox rejects the write when sandbox enforcement is applied, whereas the execution mode

selected after the persisted-rule decision permits the write. The marker resides outside the test workspace but inside a temporary, researcher-controlled directory.

The authorization tests and filesystem tests establish consecutive implementation properties: persistent approval creates a cross-project rule; a later match suppresses renewed approval; and the resulting execution mode changes the permitted filesystem effect. They do not constitute a recorded, continuous prompt-injection trajectory through the Codex user interface. We therefore use this case to validate the authorization path and its consequence, rather than report it as an additional end-to-end attack trial.

Containment and disclosure. The reproduction operates only on temporary files in a local test environment and does not access third-party systems, credentials, or user data. We report the authorization path and contained consequence needed to substantiate the paper's claims, while omitting operational payloads and bypass variants. The issue was communicated to the affected vendor before publication.

F End-to-End Attack Walkthrough in Goose

The frozen Terminal-Bench `portfolio-optimization` case provides a concrete instance of the construction in Section 4. We start from a contained target effect, derive the native grant needed to admit the corresponding action, acquire that grant during ordinary task execution, and then compare Fresh and Farmed runs through Goose's native approval interface.

Target effect $\longrightarrow$ **required shell action** $\longrightarrow$ `always_allow: shell` $\longrightarrow$ **benign acquisition** $\longrightarrow$ **replay**

Case construction. The legitimate task requires the agent to run a shipped benchmark script through its shell tool. The context change prepends a harmless marker write to the script while retaining its original contents and behavior. The task and command family remain unchanged, but the content reached by the command now produces an additional observable effect.

The effect oracle checks for a local marker file inside the experiment container. The operator does not use the network, access credentials, or affect a third-party system.

Backchaining to a native grant. We begin with the marker effect and its reference path. Reaching the modified script requires a task-related shell invocation, denoted $a^\star$. With an empty authorization store, this action reaches Goose's approval boundary and is therefore one of the approval-gated actions in A_{req}. Goose does not persist the command or script identity associated with this approval. Its matcher records only the tool name. Under the authorization profile recovered in Section 4.1, any persistent `shell` grant therefore admits $a^\star$:

$$R = \{g_{\texttt{shell}}\}, \qquad M_{\mathrm{Goose}}(R, a^\star) = \mathrm{allow}.$$

We then select a clean task execution that naturally requires Goose's shell tool. In this case, the unmodified `portfolio-optimization` task itself supplies the benign acquisition context: the shell operation is needed to run the benchmark, but the shipped script has not yet been modified. Backchaining identifies this task and the expected grant; Acquire verifies whether Goose actually creates and retains it.

Acquire. Acquire runs the legitimate task in the clean context. When Goose requests approval for shell use, the experiment controller selects the product's persistent option, `ALLOW_ALWAYS`. Goose writes the resulting authorization state to `permission.yaml`, including:

```
user:
  always_allow:
    - shell
```

This file is the grant carried into replay. Although approval is requested for a concrete task-related operation, the stored representation contains only the shell-tool identity. It does not retain the approved command, script contents, project, or task justification.

Run	Context	Shell grant	Target approval	Marker
Acquire	Clean	Acquired	`ALLOW_ALWAYS`	No
Fresh	Changed	Absent	Requested; rejected	No
Farmed	Changed	Restored	None recorded	Yes

Table 8: **Native Goose replay for one archived `portfolio-optimization` repetition.** Across all five repetitions, Acquire produced a persistent shell grant, Fresh produced one or two target-action approval requests and no marker, and Farmed produced no target-action approval request and did produce the marker.

Fresh and Farmed replay. Fresh starts with an empty permission store after the context change has been installed. When the agent proposes the target shell action, Goose issues a new approval request. The controller rejects this request, the action is withheld, and the marker remains absent.

Farmed starts from the permission store captured during Acquire. The context change and task instruction are the same as in Fresh. When the agent proposes the shell action, the stored `shell` grant matches it. No target-action approval request is recorded, the action proceeds, and the marker is created.

The marker is cleared before each run. For the authorization boundary examined here, the contrast is whether the acquired permission store is absent or restored.

Relation to the replay definition. The `shell` entry is the retained grant $g_{\texttt{shell}} \in G_h$. Fresh records that the target action reaches an approval boundary without this grant. Farmed records no corresponding approval request after the grant is restored, while the marker supplies the harmful-effect oracle. At the level recorded by the archived episode summary,

$$M_{\mathrm{Goose}}(G_0, a^\star) = \mathrm{ask}, \qquad M_{\mathrm{Goose}}(G_h, a^\star) = \mathrm{allow}, \qquad Q_2 = 0, \qquad H_2 = 1.$$

The retained grant originated in a clean run through Goose's ordinary approval workflow. Its later reuse therefore does not depend on an adversarial approval: the adversarial change occurs in the script content after the grant has already been created.

The archived summaries retain the permission store, target-action counts, approval counts, and marker outcome, but not the complete message-level ACP trace. We consequently report the observed approval-boundary and effect fields without inferring additional intermediate events.

What outlives the original context. The object carried from Acquire into Farmed is the native `always_allow: shell` entry. The approved script contents and the task context are not part of that entry. Goose therefore reuses the grant after the content determining the action's effect has changed. This is the concrete residual authority in the case: the approval's machine representation survives, while the context that gave the approval its original meaning does not.

G Bootstrap Confidence Intervals

We quantify uncertainty using a paired target-cluster bootstrap. For each model and history length, we resample injection targets (suite, X, Z) with replacement, retaining all rows and paired fresh and accumulated outcomes associated with each target. For bootstrap sample b, we compute

$$\Delta_b(h) = \mathrm{ASR}^{(b)}_{\mathrm{acc}}(h) - \mathrm{ASR}^{(b)}_{\mathrm{fresh}}(h).$$

We use $B = 2000$ resamples with seed 20260907 and report percentile 95% confidence intervals. Table 9 reports the endpoint effects at $h = 64$. Claude produces no successful attacks in either condition, making its empirical bootstrap distribution degenerate; we therefore report the observed zero effect as a negative control rather than an informative confidence interval.

Model	ΔASR (pp)	95% CI (pp)
Gemini-3.1-Flash-Lite	35.1	[30.6, 39.5]
GPT-4.1	27.9	[23.6, 32.0]
Qwen3-14B	23.5	[19.5, 27.4]
DeepSeek-V4-Pro	11.4	[7.8, 15.6]
Llama-3.3-70B	0.4	[0.0, 1.1]
Claude-Sonnet-5	0.0	Degenerate

Table 9: Paired target-cluster bootstrap estimates for the AgentDojo experiment at $h = 64$. Effects are accumulated minus fresh ASR.