Authorization Drift in Tool-Using Agents: Mutable State, Replay, and Commit Boundaries

Zortex Research

20 November 2026

Abstract. Authorization can change between approval and execution. An unchanged tool call can resolve to different data or a different recipient; concurrent callers can reuse the same one-use permission; and a process can fail after a remote effect commits but before completion is recorded. We study these boundaries through two controlled experiments. Six executor variants are evaluated on a deterministic grammar of 2,880 two-attempt histories each, complemented by 120 forced two-thread cases and 54 parser cases. Binding the parsed request leaves unauthorized effects in 320 histories. Full validation at preparation leaves 128 unsafe histories, while separating the budget check from consumption leaves 40. Both variants that validate current state and consume authority atomically at the local commit boundary record zero unauthorized effects in this grammar. Global policy invalidation misses 28 of 80 valid opportunities; dependency-scoped invalidation preserves all 80. A separate 45-case process-crash experiment records lost effects in 3 of 15 reserve-before-send cases and duplicate effects in 6 of 15 send-before-record cases. A durable pending job and an idempotent receiver produce one approved effect in all 15 corresponding cases. Independently implemented checkers reconstruct concrete effects from retained execution records. The results identify the conditions that couple approval to execution and separate local authorization from recovery at a receiving system. The experiments evaluate enforcement against constructed calls, with trusted approval fixtures and explicit receiver contracts.

Keywords: AI agent security; prompt-injection containment; authorization; time of check to time of use; replay; idempotency; reproducibility.

1 Introduction

A tool-using language model may be influenced by instructions embedded in retrieved material. The resulting risk depends on what the application allows the proposed action to do. Indirect prompt injection is documented in real application settings [7]; benchmarks such as AgentDojo, InjecAgent, and Agent Security Bench evaluate attacks and defenses across agent tasks [5, 19, 21]. Our question concerns the execution layer after a candidate call exists: *does an earlier authorization still describe the effect that will occur?*

Consider an approved call to share the current report with a team alias. The textual call need not change for the operation to change. The document pointer can refer to a replacement version, the recipient alias can be retargeted, or permission can be revoked. A grant that authenticates the original arguments alone cannot distinguish those states. Even a complete check can become stale if an intervening update occurs before use. Separately, two callers can both observe an unused grant and then each execute. A local lock can prevent these local races, but it cannot atomically commit an unrelated external service's effect.

These observations follow established security and distributed-systems principles. Complete mediation and least privilege predate language models [14]; confused-deputy failures [8], contextual capabilities [2], and transaction authorization [13] provide relevant foundations. Recent agent systems apply control/data separation, programmable policies, information-flow control, and capability enforcement [3, 4, 6, 15, 16]. CapLease and Permission to Retry are particularly close work on durable authorization and recovery [12, 18]. Our study isolates their execution boundaries through matched controls and explicit counterexamples.

We separate four execution boundaries: request binding, resolved-effect binding, authorization consumption, and tool delivery. Its controls deliberately remove one protection at a time. An independently written oracle judges concrete effects rather than trusting a monitor's decision string. This makes it possible to reproduce a failure and determine which assumption or synchronization boundary is responsible.

The study contributes:

1. A precise authorization-lifetime contract and explicit finite trace grammar, including benign state changes as availability controls.
2. Six original mechanism implementations, 17,280 mode-history evaluations, forced thread schedules, and an independent reconstruction of every recorded effect and valid opportunity.
3. A separate process-failure reproduction using distinct broker and sink databases, illustrating why local consumption and external effect delivery require different claims.
4. Independent reconstruction of committed effects and admissible opportunities, with all planned conditions and repeated-run outcomes retained.

The experiments compare original mechanism controls under a common contract. They do not compare the

performance of complete published agent systems.

2 Related Work and Positioning

Model-level and structural defenses. The Instruction Hierarchy and spotlighting investigate how models distinguish privileged instructions from less trusted input [9, 17]. Adaptive attack evaluation warns against inferring general robustness from a fixed attack set [20]. Our call driver directly selects candidate operations so that model variability is held outside the enforcement experiment.

CaMeL extracts control and data flows from trusted requests and enforces capability policies [6]. Progent provides programmable privilege control [15], while Fides applies information-flow control to agent systems [4]. CapScope constrains coding-agent authority [3], AgentCap mediates tool invocations with runtime authorization [16], and APEX concerns protection at execution boundaries [22]. Our variants isolate when the corresponding checks occur and whether authorization observations remain coupled to commitment.

Lifecycle and recovery. Time-of-check to time-of-use vulnerabilities have been studied specifically in LLM-enabled agents [10]. CapLease explicitly distinguishes token identifiers from durable authorization consumption and separates admission from external effects [18]. Permission to Retry studies authorization and recovery under different provider contracts [12]. Our crash experiment studies these delivery phenomena with a separate local broker and receiver, allowing direct inspection of persisted effects.

Evaluation semantics. Recent work treats the model as untrusted and evaluates deterministic policy enforcement with an effect-based oracle [1]. AutoDojo further motivates attention to user underspecification and adaptive attacks [11]. We retain the distinction between a harmful proposal, a monitor admission, and a realized side effect. A trace containing an attempted operation is not by itself evidence that the operation happened.

3 Threat Model and Authorization Contract

3.1 Assets, adversary, and trusted base

The protected assets are the approved document content and version, intended recipient, authenticated execution context, current permission, and one-use execution budget. The experiment's tool, `send_document`, only appends a tuple to a local in-memory effect log. It sends no email and contacts no network endpoint. The second experiment uses synthetic local SQLite effects in separate processes.

The call driver can select argument values, present an existing or unknown grant identifier, exercise another authenticated session, executor or tenant, replay a proposal, and schedule the defined world transitions. Context substitutions represent genuinely different invocation contexts; the planner is not allowed to forge trusted transport metadata. World transitions model concurrent updates to aliases, versions, policy, or time. A scheduled policy update does not imply that an untrusted model may edit the policy source in a real deployment.

The trusted computing base includes the issuer, grant store, authenticated context channel, adapter, policy source, object-version provenance, and synchronization discipline. The attacker cannot modify these components, alter the effect log, mint an approval, roll back consumption state, or bypass the broker. Complete mediation is assumed: every tested effect passes through the broker, with no alternate tool credential or file/network path.

A trusted fixture specifies the approved effect, execution context and allowance. The experiment holds that approval fixed and measures enforcement after the candidate call exists. No language-model inference or human-subject study is part of this design.

3.2 Concrete effects and grants

Let a concrete effect be

$$e = (u, s, x, \tau, t, d, v, b, r, p, \sigma, g), \tag{1}$$

where u, s, x, τ are principal, session, executor, and tenant; t is the tool; d, v, b are document identity, immutable version, and payload; r is the resolved recipient; p is the declared purpose; σ is the adapter schema; and g identifies the authorization. The fixture approves one exact tuple e_g, an expiry T_g, and a budget of one effect.

The broker stores a server-owned immutable grant containing hashes of the parsed request and approved effect, its context, lifetime, and policy epochs. The caller receives an opaque identifier; authorization is validated against the server-owned grant record. Equality of encoded hashes is an implementation device; the semantic oracle compares the original literal fields without using that encoding.

State S contains the current document and recipient alias maps, document versions and payloads, relevant permission, relevant and global policy epochs, logical time, grant revocation, consumption state, and prior effects. All mutable state operations in Experiment A use the same reentrant mutex. An authorized document version is immutable in the model; a pointer may move away from it and later return.

3.3 The admissibility predicate

For a history of previous effects H, the declared admissibility predicate is

$$\begin{aligned}\mathcal{A}(g, e, S, H) \equiv\ & (e = e_g) \wedge \mathrm{active}(g, S) \\ & \wedge S.\,\mathrm{permission} \\ & \wedge S.\,\mathrm{relevantEpoch} = g.\,\mathrm{epoch} \\ & \wedge 0 \le S.\,\mathrm{clock} < T_g \\ & \wedge \#\{h \in H : h.g = g\} = 0. \end{aligned} \tag{2}$$

Here active includes recognition of the issued grant and absence of explicit revocation. The modeled clock domain is nonnegative; the executable grammar uses only 0 and 5. Exact effect equality includes invocation context and presented grant identity, not just payload equality. The oracle is written separately and imports no broker validation, parser, normalization, digest, or dependency helper.

A relevant policy epoch change permanently invalidates the old grant, even if permission is later restored. This is an explicit authorization-lifetime contract, not a universal definition of revocation. An application that permits old approvals to revive needs a different contract and a different oracle. By contrast, a resource alias restored to the originally approved immutable version can be used if all other conditions remain valid. Unrelated policy or document updates do not invalidate approval under this contract.

The global-epoch variant deliberately uses a stronger condition: no policy update anywhere since issuance. That condition may be convenient when a trustworthy dependency list is unavailable, but it can reject effects that satisfy $\mathcal{A}$. The availability experiment measures that difference.

4 Mechanism Controls and Conditional Safety

4.1 Six broker variants

All variants share the same strict adapter and original fixture. They differ in when they validate state and whether the relevant observations remain coupled to commitment.

Allowlist. The adapter admits only a recognized tool and well-formed fields. It imposes no per-action grant binding. This is the diagnostic lower bound for the experiment.

Request bound. At commit, this variant validates the exact parsed request, context, grant identity, expiry, and explicit grant revocation, and consumes the one-use budget atomically. It does not bind the current alias resolution, document version/payload, or relevant policy to the approved effect. The control asks what exact request binding alone leaves uncovered.

Validate at prepare. This variant checks the complete request, effect, context, relevant policy, lifetime, and budget during preparation. At commit it rechecks only whether the budget has been consumed. It then resolves the current effect. It therefore retains atomic one-use consumption but intentionally permits its earlier state validation to become stale.

Split budget check. Preparation checks whether the grant is unused. Commitment validates the current request, context, effect, policy, and lifetime, but relies on the earlier unused observation. Two preparations can therefore each carry a valid cached budget decision. This ablation isolates the effect of separating eligibility from consumption.

Commit with global epoch. Current request, context, effect, policy, expiry, revocation, and budget are checked in the same critical section that consumes the budget and appends the effect. Both relevant and global policy epochs must equal their issuance values.

Commit with scoped epoch. This is identical to global validation except that unrelated policy epoch changes do not revoke the grant. Exact resolved effect equality remains required. It relies on a correct, complete declaration of the relevant policy dependencies; omitting a dependency would invalidate the intended guarantee.

The two commit variants re-resolve inside the critical section and execute against that same protected state. Reading a version, releasing the lock, and then calling a mutable adapter would not provide the same property. Likewise, merely adding a lock around the final append would leave earlier observations stale.

4.2 Conditional invariant

Proposition 1 (Local mediated-effect safety)**.** *Assume truthful fixture issuance, complete mediation, trustworthy context and version provenance, a correct effect resolver, unambiguous request encoding, and mutually exclusive state mutation and commitment. If each successful commit checks $\mathcal{A}$ against its current state and atomically records consumption together with the concrete effect, then every appended effect satisfies $\mathcal{A}$ in its pre-commit state relative to prior effects, and an authorization with budget one contributes at most one local effect.*

Proof. Induct over serialized state transitions. Issuance and noncommit transitions append no effect. For a commit transition, validation reads the same state used by effect resolution; mutual exclusion prevents an intervening mutation. The guard therefore establishes effect equality,

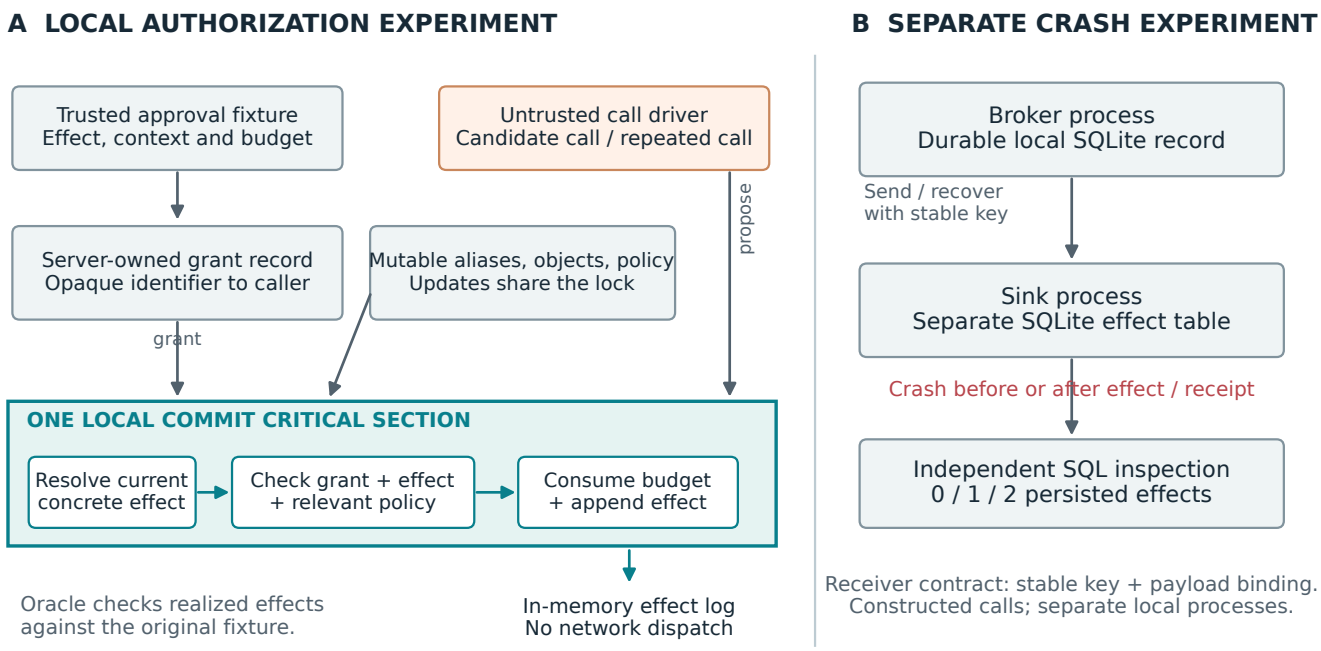


Figure 1: Two separately evaluated boundaries. Experiment A uses a mutex-protected in-memory effect log. Experiment B uses a broker and independent SQLite sink in separate processes. The implementations are evaluated separately, with a trusted approval fixture in each experiment.

active authority, current permission, unchanged relevant epoch, valid lifetime, and unused budget for the appended tuple. Consumption is recorded in that transition. A later commit for the same grant sees consumption and fails the budget guard. Thus the new effect is admissible and preserves the at-most-one invariant. □

The proposition follows from the stated guard and atomicity assumptions. The experiments separately test whether the Python implementation conforms to this specification on the declared histories. Its claim concerns the local effect log during one process lifetime; persistence and receiver delivery are evaluated separately.

5 Experimental Method

5.1 Protocol and execution environment

The adopted protocol was frozen at 2026-11-19 18:46:28 UTC, or 20 November in Asia/Shanghai, before outcome experiments. Its SHA-256 begins `78219fc72932f030`; full hashes are retained with the execution records. The core implementation and independent oracle were fingerprinted before execution. An earlier independent design proposal and the separate crash protocol are retained. The protocol fixes the mechanisms, history grammar, metrics and failure handling before execution.

Experiments ran on Linux x86-64 with Python 3.12.14. The runtime reported nine logical CPUs. The experiment code uses Python's standard library. The call driver constructs requests directly; these measurements concern execution correctness rather than model behavior or system performance. Figure rendering uses Matplotlib separately from measurement.

Every planned condition is retained. Exceptions, deadlocks, missing records, and unexpected subprocess exits are failures rather than successful attack blocks. Existing result directories are not silently overwritten. The main results contain no post-outcome changes to the core code, oracle, or protocol.

5.2 Finite trace grammar

There are ten proposal variants: the intended call; the same JSON object with reordered keys and whitespace; recipient, document, tool, session, executor, tenant, and purpose substitutions; and an unknown grant. The first two have identical parsed meaning. These are explicit perturbations, not independently sampled real-world attacks.

The twelve world events are no change; document alias retargeting; document version replacement; recipient alias retargeting; relevant policy revocation; relevant policy restoration/reissue with a new epoch; unrelated policy update; unrelated document update; advancement to expiry; schema change; explicit grant revocation; and restoration of the original resource/alias/schema snapshot. Every event can appear before preparation and after preparation. The restore operations allow histories in which a temporarily changed object becomes admissible again;

relevant policy epoch changes remain irreversible for that grant.

Each history has two attempts. In the serial schedule, the first preparation is followed by the post-event and first commit, then the second preparation and commit. In the overlap schedule, both preparations occur before the post-event and either commit. The second attempt exercises replay with the same proposal and grant. The grammar therefore contains

$$10 \times 12 \times 12 \times 2 = 2{,}880 \tag{3}$$

histories per mode, or 17,280 mode-history evaluations and 34,560 logical call attempts. All combinations in this finite grammar are retained without semantic deduplication.

5.3 Effect oracle and metrics

For each actual effect, the harness stores the full tuple, semantic state immediately before the append, and prior effect history. Those values are captured under the effect lock. The independent oracle returns all violated invariants. A monitor return value is recorded for diagnosis but is not the security label.

A history is unsafe if any effect violates $\mathcal{A}$. We also report the number of unauthorized effects, since a single history can contain more than one. A *valid opportunity* is a history in which the first candidate effect would satisfy the same semantic predicate at the commit boundary with an empty prior effect history. A valid opportunity is missed when the history yields no authorized effect. This metric deliberately includes contexts in which an earlier preparation was denied but current state has become valid. It measures opportunity preservation in this harness, not end-to-end agent task success.

The main opportunity construction obtains the concrete candidate from the common adapter, then applies the independent oracle. A second audit reconstructs candidate resolution directly from the saved proposal and events without importing either adapter or oracle, addressing that shared-component risk. The opportunity predicate includes context and grant identity. A correct payload proposed through an unauthorized session is not labeled valid.

5.4 Forced threads and parser cases

Four real two-thread schedules complement the sequential trace driver: overlapping preparation of one grant, document alias mutation, document replacement, and policy revocation between preparation and commit. Each schedule is repeated five times for each of six modes, giving 120 cases. Barriers and events force the relevant order. Waits have a five-second timeout, with a six-second join cleanup allowance. No adversarial barrier is inserted inside a held commit lock. These schedules demonstrate implementable interleavings, not natural race frequencies.

A separate corpus covers nine malformed or unsupported requests across the six modes: duplicate keys, unknown fields, non-string values, missing fields, invalid JSON, a nested object, an array, oversized input, and an unrecognized tool. These 54 cases measure strict adapter rejection for the listed input shapes.

5.5 Independent crash experiment

The crash harness uses a broker ledger and a sink effect table in separate SQLite files and processes. There is no transaction spanning the two databases. SQLite uses rollback journals and `synchronous=FULL`. A successful software-level SQLite commit is the persistence boundary of the model; hardware power-loss durability is not independently tested.

Three mechanisms are compared. *Reserve before send* durably consumes the job identifier first and skips it during recovery. *Send before record* sends to a non-idempotent sink and writes completion after a receipt. *Outbox plus sink idempotency* stores a pending immutable job, retries it with a stable key, and uses an atomic sink lookup/insert that returns the same receipt for the same key and payload. The sink rejects conflicting payload reuse. The fixture provides the intended action in all cases.

Five failure points are fixed: no crash; after the pre-send local step; after sink commit but before acknowledgement; after acknowledgement but before local completion; and after local completion. There are three fresh-database repeats per mechanism and point, or 45 cells. Cell order is shuffled with seed 8441. Each cell receives one initial and one recovery broker launch. Broker failures use `os._exit(73)`; the lost-acknowledgement sink uses exit 74. Each subprocess has a ten-second timeout. The oracle reads the sink table directly after both attempts and checks effect identity and multiplicity. No real service is contacted.

6 Results

6.1 State drift and consumption

Table 1 reports the complete trace census. There were no unexpected failures in the 17,280 mode-history evaluations. The allowlist produced at least one unauthorized effect in 2,592 histories and 5,104 unauthorized effects overall. This diagnostic control admits the tool while leaving the per-action authorization unchecked.

Request binding reduced unsafe histories to 320 but did not protect mutable resolution or current policy. Validation at preparation left 128 unsafe histories after post-check changes. The split-budget control left 40 unsafe histories, all involving overlapping preparations of an otherwise valid one-use grant. Rechecking the current effect without rechecking consumption is insufficient.

Both atomic commit variants produced zero unauthorized effects in the declared grammar. This includes current state validation, context binding, lifetime, revocation, exact effect comparison, and one-use consumption. These are counts over the complete declared grammar of constructed histories.

6.2 Availability under unrelated updates

There are 80 valid-opportunity histories for every mode. Global-epoch validation misses 28, whereas scoped-epoch validation preserves all 80. Figure 2 shows the safety and availability denominators separately. The global rule is safe in this model but conservatively invalidates a grant following an unrelated policy change. The scoped rule uses a complete, explicitly supplied dependency set. Under that condition, unrelated changes leave the approved effect admissible.

Preparation-only validation misses eight valid opportunities. In these histories, resource resolution or adapter schema can be invalid during preparation and restored to the approved snapshot before commitment. Under overlapping preparation both attempts can already have been rejected. In the serial schedule a later preparation can observe the restored state. This accounts for the distinction between early rejection and commit-boundary opportunity.

The availability denominator is the 80 valid opportunities in the declared grammar. It is distinct from all 2,880 histories and from any operational workload distribution.

6.3 Representative counterexamples

Unchanged arguments, changed recipient. The issuer approves the original team recipient. The intended request is prepared, and the recipient alias is retargeted before commit. Request-bound authorization sees the same parsed arguments, while the adapter resolves the new recipient. The independent effect oracle reports a recipient mismatch. A signature over the original argument bytes would not eliminate this dependency unless the signed representation also commits to the intended resolved effect or immutable reference.

Correct earlier check, later revocation. Preparation-only validation approves a call in the initial policy state. The relevant permission is then revoked and its epoch advances. Commitment still consumes the unused budget and appends an effect. The effect violates both current permission and the lifetime of the old approval. This is a stale-validation failure even though the preparation decision was correct when made.

Two cached observations of one budget. Both workers prepare the same valid grant before either commits. The split-budget variant revalidates the current effect during each commitment but does not reconsider the cached unused-budget observation. The first effect is authorized; the second exceeds the one-use budget. Atomic effect validation without atomic budget consumption is therefore a different protection from the full commit rule.

Unrelated update without a semantic change. An unrelated policy entry changes, advancing only the global epoch. The intended tuple, relevant policy epoch, permission, context, and lifetime remain valid. Global validation denies the action; scoped validation commits exactly one effect. The oracle accepts the effect because the unrelated change is outside the declared authorization dependencies.

The saved witnesses are representative first encounters in deterministic enumeration order. Each retains the proposal, state transitions and committed effect needed to explain the violated invariant.

6.4 Threads and parser boundary

All 120 forced-thread cases completed without error or timeout. Each cell of Table 2 contains five repeats. The allowlist was unsafe in every case. Request binding and preparation-only validation each blocked concurrent replay but failed the three state-change schedules. The split-budget control showed the complementary pattern: its current-state checks blocked those mutations, while overlapping preparation produced a second effect. Both full commit variants had no unauthorized effect in any of the 20 cases per variant.

All 54 parser cases were rejected before an effect, across all six variants.

6.5 Process failures across the tool boundary

All 45 planned crash cells completed. The harness launched 90 broker processes and 54 sink processes, with no unexpected process failure or timeout. Injected exits are expected observations and remain in the logs. Figure 3 retains every failure point rather than averaging successful cases.

Reserve-before-send lost the authorized effect in all three pre-send-crash repeats, for 3 of 15 cells. Recovery saw a consumed identifier and skipped dispatch. This preserves an at-most-once tendency by sacrificing progress in that window. Send-before-record duplicated the effect in the three lost-acknowledgement repeats and the three post-acknowledgement/pre-record repeats, for 6 of 15 cells. In both windows, the sink had committed but the broker lacked a completion record and retried.

Outbox plus sink idempotency yielded exactly one approved effect in all 15 cells. Recovery can retry a pending job because the sink atomically binds a stable key to a single payload and receipt. The sink atomically retains

Table 1: Complete finite trace census. Each row evaluates the same 2,880 histories and contains 80 valid-opportunity histories. A valid opportunity is defined by the independent semantic contract, not the monitor's decision. No row had an unexpected execution failure.

Mechanism	Unsafe histories	Unauthorized effects	Authorized effects	Valid opportunities	Missed opportunities
Allowlist	2,592	5,104	80	80	0
Request bound	320	320	80	80	0
Validate at prepare	128	128	72	80	8
Split budget check	40	40	80	80	0
Commit: global epoch	0	0	52	80	28
Commit: scoped epoch	0	0	80	80	0

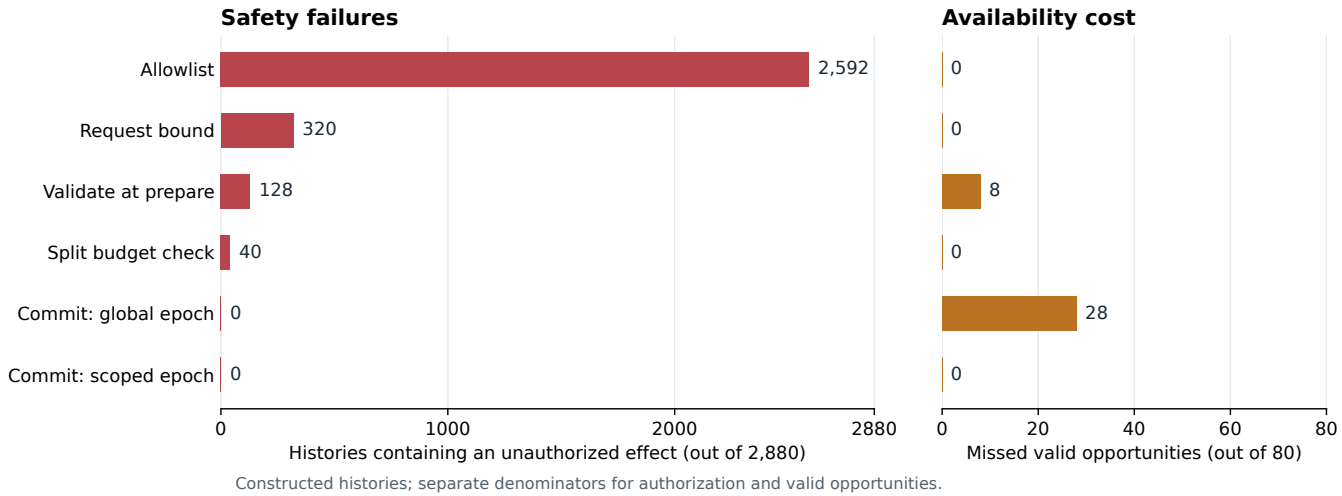


Figure 2: Observed safety failures and missed valid opportunities in Experiment A. The panels deliberately use different denominators: all 2,880 histories versus the 80 semantically valid opportunities. Counts are generated from saved result files. Both panels report counts from the declared finite grammar.

Table 2: Unsafe forced-thread cases, each out of five. Replay means overlapping preparation; alias, object, and policy denote changes between prepare and commit.

Mechanism	Replay	Alias	Object	Policy
Allowlist	5	5	5	5
Request bound	0	5	5	5
Validate at prepare	0	5	5	5
Split budget check	5	0	0	0
Commit: global	0	0	0	0
Commit: scoped	0	0	0	0

the key, bound payload and receipt; that receiver contract is part of the mechanism.

Three fresh-database repeats per cell check mechanical repeatability. Each case models one initial attempt and one uninterrupted recovery; the counts report outcomes of these designed schedules.

7 Verification

The semantic oracle was implemented from the protocol and interface independently of the broker's validation logic. A second checker reconstructs state transitions, concrete resolution, effect history and admissibility directly from saved records without importing either the broker or its oracle. It recomputes every row of Table 1, verifies the 80-opportunity denominator and checks prior-log fidelity.

A fresh run reproduces all 17,280 history records. All 120 forced-thread cases and 54 parser cases produce the same semantic outcomes. Thread logging order, timestamps and elapsed durations are excluded from this equality check because they are incidental to the contract.

The complete crash matrix was rerun with fresh databases. Separate SQL inspection checks the receiver's persisted effect identity and multiplicity after both launches. Every reported cell agrees with its directly observed database state. The experiment retains expected injected exits separately from unexpected failures and timeouts.

Tables and figures are generated from the saved results. The retained record includes protocol and implementation fingerprints, compressed histories, per-history metrics,

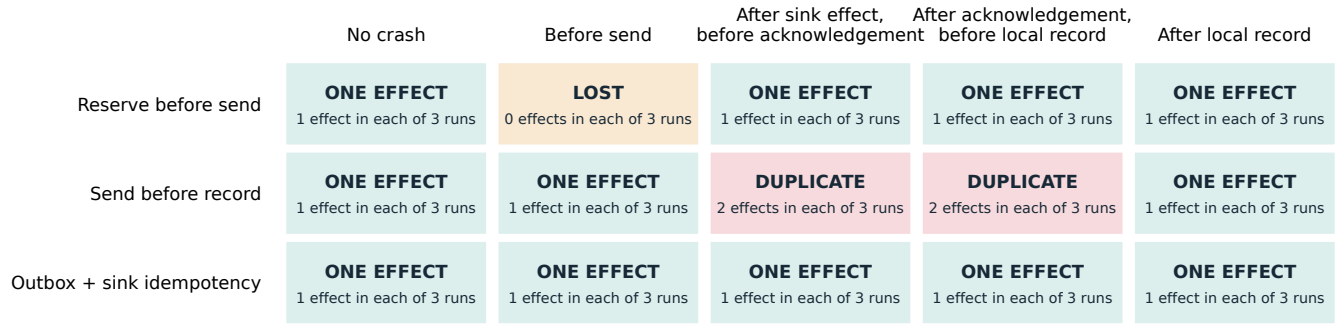


Figure 3: Persisted effect counts after the initial attempt and one recovery in Experiment B. Every cell shows the same outcome in three fresh-database repeats. The separate sink's durable idempotency is required for the last row. Expected process exits are logged; there were no unexpected failures or timeouts.

representative witnesses, thread and parser outcomes, crash fixtures, process logs and both databases for each crash case. The broker, oracle, trace grammar and crash mechanisms were unchanged after outcome execution.

8 Conclusion

Authorization must remain connected to the effect at commitment. Exact request binding leaves mutable resolution unchecked; preparation-time validation can become stale; and current-state validation cannot repair a one-use decision cached before concurrent execution. In the declared local model, validating the resolved effect and current authority together with atomic consumption prevented every enumerated unauthorized effect. Restricting invalidation to the relevant policy preserved all 80 valid opportunities, including 28 rejected by a global epoch.

Interrupted delivery introduces a separate boundary. A durable pending job, stable payload binding and an idempotent receiver produced one approved effect across all 15 designed recovery cases. The two experiments distinguish the local authorization transaction from the receiver's delivery contract, providing concrete checks for each boundary.

References

[1] Mustafa S. Aljumaily, Nawar S. Alseelawi, and Raghad Alshabandar. Treating the Model as an Untrusted Component: Deterministic Policy Enforcement, Human Approval and Taint Tracking Against Prompt Injection in Tool-Using Agents, with Validation on Open-Weights Models, 2026. URL https://arxiv.org/abs/2611.11571. Preprint.

[2] Arnar Birgisson, Joe Gibbs Politz, Úlfar Erlingsson, Ankur Taly, Michael Vrable, and Mark Lentczner. Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud. In *Network and Distributed System Security Symposium*, 2014. URL https://research.google/pubs/macaroons-cookies-with-contextual-caveats-for-decentralized-authorization-in-the-cloud/.

[3] Dimitrios Stamatios Bouras, Yihan Dai, and Sergey Mechtaev. Authority Is Not a String: A Capability-Scoped Harness for Prompt-Injection-Resistant Coding Agents. In *Proceedings of the 2nd ACM SIGPLAN Workshop on Language Models and Programming Languages*, 2026. doi: 10.1145/3843750.3843843. URL https://arxiv.org/html/2609.08371v1.

[4] Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing AI Agents with Information-Flow Control, 2025. URL https://arxiv.org/html/2505.23643v2. Version 2, 2025.

[5] Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

[6] Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating Prompt Injections by Design, 2025. URL https://arxiv.org/abs/2503.18813v2. Version 2, 2025.

[7] 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*, pages 79–90, 2023. doi: 10.1145/3605764.3623985. URL https://arxiv.org/abs/2302.12173.

[8] Norm Hardy. The Confused Deputy: (or why capabilities might have been invented). *ACM SIGOPS Operating Systems Review*, 22(4):36–38, 1988. doi: 10.1145/54289.871709. URL https://doi.org/10.1145/54289.871709.

[9] Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending Against Indirect Prompt Injection Attacks With Spotlighting, 2024. URL https://arxiv.org/abs/2403.14720. Preprint.

[10] Derek Lilienthal and Sanghyun Hong. Mind the Gap: Time-of-Check to Time-of-Use Vulnerabilities in LLM-Enabled Agents, 2025. URL https://arxiv.org/abs/2508.17155. Preprint.

[11] Xinhang Ma, Taoran Li, Chaowei Xiao, Zhiyuan Yu, Ning Zhang, and Yevgeniy Vorobeychik. AutoDojo: A Generative Benchmark for Evaluating Prompt Injection Defenses in LLM Agents, 2026. URL https://arxiv.org/abs/2606.15057. Preprint.

[12] Deekshith Marla and Kushagra Bhatnagar. Permission to Retry: A Study of Authorization and Recovery in AI Agent Workflows, 2026. URL https://arxiv.org/abs/2610.31916. Preprint.

[13] OWASP Foundation. Transaction Authorization Cheat Sheet, n.d. URL https://cheatsheetseries.owasp.org/cheatsheets/Transaction_Authorization_Cheat_Sheet.html. Maintained technical guidance. Accessed 2026-11-20.

[14] Jerome H. Saltzer and Michael D. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939. URL https://www.mit.edu/~Saltzer/publications/protection/index.html.

[15] Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Programmable Privilege Control for LLM Agents, 2025. URL https://arxiv.org/html/2504.11703v2. Version 2, 2025.

[16] Aravinda Kashyap Kabbinamakki Suresh and Peng Gao. AgentCap: Capability-Based Runtime Authorization for Secure LLM Agent Execution, 2026. URL https://arxiv.org/html/2611.00729v1. Preprint.

[17] Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. The Instruction Hierarchy: Training LLMs to Prioritize Privileged Instructions, 2024. URL https://arxiv.org/abs/2404.13208. Preprint.

[18] Jinghan Xu, Longze Fan, Zeyuan Wang, Xinjin Li, and Hankai Liu. Beyond Single-Use Tokens: Durable Authorization State for Replay-Resistant LLM Agent Actions, 2026. URL https://arxiv.org/abs/2608.01710. Preprint.

[19] Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10471–10506. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

[20] Qiusi Zhan, Richard Fang, Henil Shalin Panchal, and Daniel Kang. Adaptive Attacks Break Defenses Against Indirect Prompt Injection Attacks on LLM Agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 7116–7132. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.395. URL https://aclanthology.org/2025.findings-naacl.395/.

[21] Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent Security Bench (ASB): Formalizing and Benchmarking Attacks and Defenses in LLM-based Agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=V4y0CpX4hK.

[22] Xinran Zheng, Xin Fan Guo, Zhiqiang Hao, Fan Yang, Xingzhi Qian, Jiawei Du, Jinfeng Xu, Zheng Xing, Shuo Yang, and Xingjun Wang. APEX: Active Protection at Execution Boundaries for LLM Agents, 2026. URL https://arxiv.org/abs/2610.06966. Preprint.