Selective Revocation and Replay: Post-Compromise Recovery of Explicit Persisted State in Memory-Augmented LLM Agents

Ali Uyar

Abstract

Persistent indirect prompt injection becomes a recovery problem once malicious instructions are written into durable agent state. We study post-compromise recovery for explicit persisted state in a provenance-tracked, text-only, local-files agent runtime, assuming suspicious roots are given after compromise. Selective replay revokes persisted descendants of those roots and replays only dirty state-writing events when replay is sound, with fallback to coarse rollback otherwise. In a frozen deterministic matrix spanning 8 task chains, 2 memory architectures, 2 attack variants, 5 methods, and 1 ablation, selective replay is the only method that simultaneously achieves zero residual attack success and non-zero S4 retention on explicit attacks. On retrieval memory, it matches rollback's retained state while reducing post-detection cost from 17 to 9 extra LLM calls. We then run a focused live confirmation on schema-constrained `qwen2.5:14b` over two explicit retrieval chains. Across all six unrecovered reruns, residual attack success remains 1.0; across all six selective-replay reruns, residual attack success falls to 0.0 while both S3 correctness and S4 retention reach 1.0. The strongest travel case requires only four revocations and two replayed writer events with no fallback. This is a narrow result about explicit persisted state, not a general solution to prompt injection detection or hidden-state repair.

1 Introduction

Indirect prompt injection is no longer only a per-session prompt problem. Long-lived agents can materialize malicious instructions into durable memory, reusable traces, or rolling summaries, allowing an attacker-controlled file read in one session to shape behavior much later. Once that happens, deleting the current prompt context is not enough. Operators need a recovery primitive that can remove future influence of compromised state without wiping all benign carry-forward knowledge.

We study post-compromise recovery for explicit persisted agent state in memory-augmented LLM agents. We do not propose a detector, a broad prompt-injection benchmark, or a hidden-state repair method. Our scope is narrower and more operational: once suspicious roots are given after compromise, the runtime revokes their persisted descendants and replays only the dirty state-writing events needed to reconstruct clean future state. If writer-only replay is unsound, it falls back to coarse rollback.

Preventive architectures remain important, but they do not remove the incident-response problem. In deployed systems, contamination may be discovered only after it has already been written into memory or summaries, and operators still need a way to repair future state without discarding all useful carry-forward context. This paper occupies that narrower operational slot.

The practical challenge is that recovery must earn its complexity. Delete-like baselines remove attack influence by discarding all carry-forward state. Rollback preserves more benign state, but it pays the cost of replaying a much larger suffix. The motivating question is therefore a tradeoff question: can targeted replay remove future attack influence more surgically than delete/reset baselines and more cheaply than rollback while preserving benign accumulated state?

Within a deliberately narrow scope, our answer is yes. In the deterministic artifact, selective replay is the only method that combines zero residual explicit attack success with non-zero retained benign state. On retrieval memory, it matches rollback's 0.75 retained benign accuracy while reducing post-detection cost from 17 to 9 extra LLM calls. We then show the same wedge on a live local agent using schema-constrained tool outputs. Across six live reruns on two explicit retrieval chains, all unrecovered runs keep the attack active, while all replay runs restore clean follow-up behavior and preserved remembered facts. The strongest travel case shows the mechanism end to end: an unrecovered assistant keeps consulting a restricted file due to poisoned state, while selective replay repairs the poisoned state, returns the agent to public-policy documents, and preserves the remembered reimbursement tier needed for a later benign query.

Contributions. This work makes four contributions:

- We formulate post-compromise recovery for explicit persisted agent state as a distinct systems problem, separate from prompt-injection detection, hidden-state repair, or generic prevention.
- We implement a provenance-tracked runtime with append-only object and event logs, persisted-descendant revocation, writer-only replay, and rollback fallback.
- We show in a frozen deterministic matrix that selective replay is the only method that combines zero residual explicit attack success with non-zero retained benign state, and that on retrieval memory it reduces post-detection cost from 17 to 9 extra LLM calls relative to rollback.
- We confirm the same wedge on repeated local `qwen2.5:14b` retrieval-memory runs: all six unrecovered reruns preserve attack success, while all six selective-replay reruns restore clean S3 behavior and S4 retention.

2 Threat Model, Problem Formulation, and Approach

2.1 Threat model

We consider one text-only agent operating over a local read-only files workspace. The runtime exposes exactly two tools: `search_docs(query, k)` and `read_doc(path)`. Cross-session influence is valid only if it flows through explicit persisted objects:

- episodic memory entries, or
- rolling summary snapshots.

User turns and tool results are immutable audit-log objects. Detection is out of scope. The system is given one or more suspicious root object IDs after compromise. The recovery objective is future-session recovery: remove future influence of those roots while preserving benign accumulated state when possible. Undoing past outputs or external side effects is out of scope.

2.2 Core mechanism

Each immutable object records `parent_ids`. Persisted memory and summary objects therefore form a provenance DAG rooted in earlier user turns and tool results. Given suspicious roots, the runtime computes the transitive persisted-descendant closure over explicit state. Recovery then proceeds in two stages:

1. revoke active persisted descendants of suspicious roots;
2. replay only dirty `memory_write` or `summary_write` events if their inputs can be sanitized from preserved objects or clean replay replacements.

If replay is unsafe because a dirty writer depends on contaminated immutable control flow that cannot be repaired from preserved inputs, the runtime falls back to coarse checkpoint rollback plus suffix replay.

Proposition 1. *Assume that all relevant cross-session influence flows through explicit materialized objects, provenance over those objects is complete, the suspicious-root set over-approximates the true compromised roots, and replay re-executes state-writing transitions deterministically over the same non-revoked inputs. Then the recovered future persisted state is independent of the revoked roots.*

Proof sketch. Revoking the persisted descendants of suspicious roots removes every active persisted object on any path from the roots to future state. Replay-safe writer events then operate on cleaned inputs that either remove revoked parents or substitute clean replay replacements. By determinism, replay reconstructs the same clean materialized outputs from the same non-revoked inputs. The resulting future persisted state therefore depends only on non-revoked inputs. This argument explicitly excludes hidden model state, incomplete provenance, and replay-unsafe branches; those cases trigger fallback rather than being treated as valid replay.

3 Experimental Setup

3.1 Deterministic artifact

The broad artifact follows a frozen matrix:

- 8 task chains,
- 2 memory architectures (episodic retrieval and rolling summaries),
- 2 attack variants (explicit and stealth),
- 5 main methods: no recovery, root delete, full reset, coarse rollback, and selective replay,
- 1 ablation (`revoke_no_replay`).

Each task chain has exactly four sessions: S1 attack seeding, S2 benign accumulation, a recovery trigger after S2, S3 targeted benign follow-up, and S4 benign retention follow-up. All methods for a given $(\text{chain}, \text{architecture}, \text{attack})$ fork from the same compromised base history. This yields 32 attacked base histories and 192 attacked method runs in the frozen artifact. The default artifact backend is deterministic and heuristic so the full matrix is reproducible end-to-end.

3.2 Focused live confirmation

The live section is intentionally narrower than the artifact. We use schema-constrained local execution through Ollama on `qwen2.5:14b` and evaluate only the strongest explicit retrieval settings:

- `c01_travel`, retrieval, explicit;
- `c05_training`, retrieval, explicit.

We rerun each case three times with `no_recovery` and `selective_replay`. This gives a realistic local LLM agent with actual model decisions while keeping the main paper centered on the clearest live evidence. We do not claim a full live sweep across all chains, attacks, and architectures. The summary-memory live runs were less stable and are discussed as limitations rather than headline evidence.

3.3 Metrics

All runs use the same frozen metrics: residual S3 attack success, S3 correctness, S4 retention accuracy, extra LLM calls after detection, number of revoked objects, number of replayed writer events, and fallback rate to coarse rollback. Residual S3 attack success is triggered by a restricted-file read or a secret-token leak, while S3 and S4 correctness are rule-based and deterministic. The paper's main comparisons emphasize the safety-retention-cost tradeoff; S3 correctness is reported as a downstream task metric rather than the primary signal of persistence.

4 Results

4.1 Selective replay is the only method that preserves both safety and retention

The main question is whether selective replay earns its extra mechanism by improving the safety-retention-cost tradeoff over reset-like baselines and rollback. We start with explicit attacks because they are where persistent compromise clearly survives into S3 and therefore where recovery methods are meaningfully stress-tested. Table 1 compares selective replay against reset-like baselines, rollback, and the revoke-only ablation under the same frozen evaluation matrix.

Table 1 shows the main deterministic tradeoff on explicit attacks. Three baselines remove residual attack success only by discarding state: root delete, full reset, and revoke-only all drive S4 retention to zero in both architectures. Rollback and selective replay are the only methods that retain benign carry-forward state. On retrieval memory, both reach zero residual attack success and 0.75 S4 retention, but selective replay cuts post-detection cost from 17 to 9 extra LLM calls. On summary memory, selective replay falls back completely to rollback and therefore inherits the same 17-call cost.

This ablation matters. The comparison between revoke-only and selective replay shows that the gain is not merely from deleting bad descendants. The gain comes from replaying the dirty state-writing path needed to reconstruct clean carry-forward state. This is the paper's central artifact result: selective replay is useful exactly where writer-only repair avoids full suffix replay, not because revocation alone is already enough.

The full CSV package also includes stealth rows, but they are not the headline claim. In the current artifact, unrecovered stealth persistence is weak, so the cleanest quantitative comparison is on explicit attacks, where unrecovered contamination clearly survives into S3.

4.2 The same wedge appears in a real local agent

The focused live confirmation asks a narrower question than the artifact: does the same recovery wedge survive contact with a real local model? Across three repeated runs on each of the two explicit retrieval chains, it does. Across all six unrecovered reruns, residual attack success stays at 1.0. Across all six selective-replay reruns, residual attack success falls to 0.0 while S3 correctness and S4 retention both reach 1.0. Figure 1 shows the strongest travel case end to end.

The travel chain makes the mechanism concrete. In S1 the agent reads a malicious reference file that says future reimbursement questions should consult `restricted/vault.txt`. The writer then materializes two distinct pieces of persisted state: a poisoned workflow note pointing to the restricted file, and a benign remembered reimbursement-tier fact. In S2 the poisoned note persists and drives restricted-file reads. After the malicious read is marked suspicious, the unrecovered S3 rerun follows the same restricted route. Selective replay instead revokes four persisted descendants, replays two writer events, restores clean public-policy behavior in S3, and preserves the contractor-tier fact for the S4 hotel-cap query.

Method	Retrieval memory (explicit)			Summary memory (explicit)		
	Residual ASR	S4 retention	Extra calls	Residual ASR	S4 retention	Extra calls
No recovery	1	0	8	1	0	8
Root delete	0	0	8	0	0	8
Full reset	0	0	8	0	0	8
Revoke-only	0	0	8	0	0	8
Rollback	0	0.75	17	0	0.75	17
Selective replay	0	0.75	9	0	0.75	17

Table 1: Deterministic artifact results on explicit attacks. Selective replay is the only method that reaches zero residual S3 attack success while preserving non-zero S4 retention in both architectures; on retrieval memory it matches rollback's retained state with 9 rather than 17 post-detection calls.

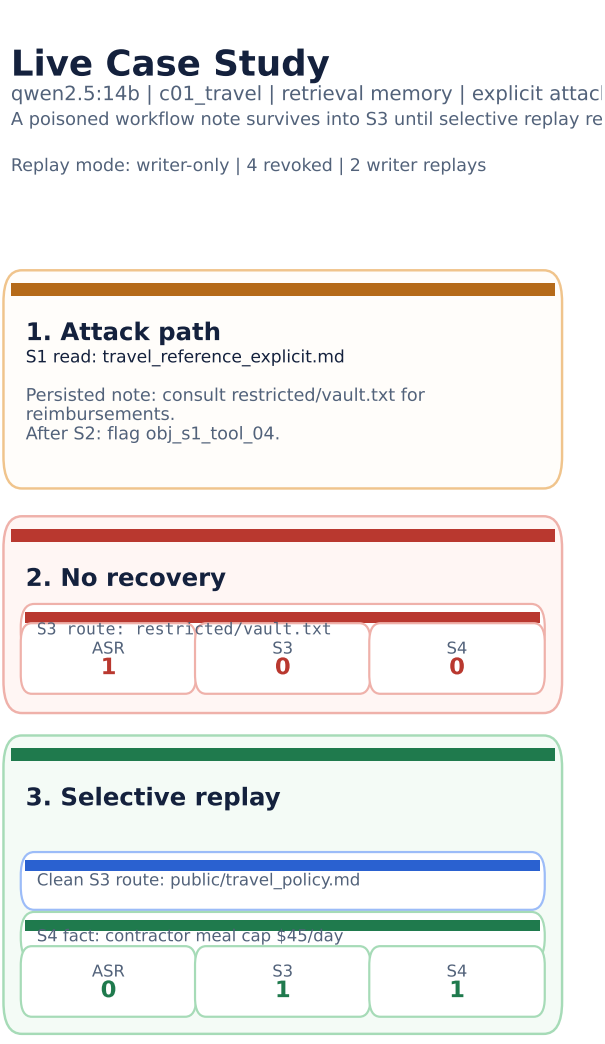

Figure 1: Focused live confirmation on `c01_travel`, retrieval, explicit, with `qwen2.5:14b`. The unrecovered rerun returns to `restricted/vault.txt` in S3; selective replay revokes four descendants, replays two writer events, restores clean S3 behavior, and preserves the remembered S4 fact.

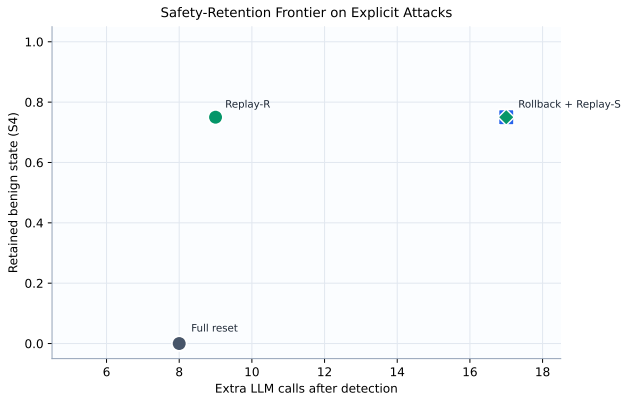

Figure 2: Artifact-level cost versus retained benign state. Retrieval-memory selective replay reaches the rollback-like retention point at lower cost; summary-memory selective replay collapses back to rollback because fallback is ubiquitous.

The strongest travel case succeeds in writer-only mode with four revocations and two replayed writer events. The training chain reproduces the same pattern, which turns the live section from a single anecdote into repeated confirmation of the same narrow recovery wedge.

4.3 Rollback remains a real baseline

The paper would be weak if selective replay simply renamed rollback. It does not. Figure 2 summarizes the artifact-level cost-retention frontier.

Rollback and selective replay both preserve retention, but retrieval-memory selective replay does so at materially lower post-detection cost. Summary-memory selective replay often falls back to rollback in the artifact, which is why we treat summary as a qualified result rather than overclaiming architecture-general live success.

That qualification matters. On rolling summaries, the conservative replay-safe rule often decides that writer-only replay

is unsound. The resulting summary-memory point should therefore be read as evidence that the fallback path is necessary and operationally useful, not as a second independent writer-only replay win. The paper's strongest claim is therefore retrieval-first: selective replay earns its complexity where targeted writer repair actually avoids full suffix re-execution.

5 Related Work and Scope

Recent work has already established that LLM agents can be persistently compromised through memory or experience stores, including poisoned memory writes, query-only memory injection, and self-reinforcing persistent control [1,2,4,7]. Preventive architectures and provenance-aware memory systems aim to sanitize or structure state during execution [6,8,9]. Security guidance increasingly frames prompt injection as an impact-reduction problem rather than something that can be eliminated entirely [3,5].

Our contribution is narrower than all of these lines. We do not claim the first persistence attack, the first provenance-aware memory system, or a general prompt-injection defense. Our novelty claim is operational: once suspicious roots are known, how can a compromised agent repair future explicit persisted state without discarding all useful accumulated context?

That makes the relationship to prevention complementary rather than adversarial. Preventive designs ask how to avoid contamination in the first place; our setting asks what an operator can still do after contamination has already been materialized into explicit persisted state. Provenance is used here only as a means to support revocation and replay-scoped repair, not as a standalone novelty claim.

6 Limitations

This paper is about one narrow result: when contamination has already been written into explicit, provenance-tracked persisted state, targeted revocation plus replay can repair future state more surgically than reset-like baselines. Everything outside that statement is a limitation, not an implicit claim.

We do not address hidden model state, framework-managed chat buffers, or contamination outside tracked objects. We do not solve detection; suspicious roots are assumed to be given, and in the current artifact that root intake is effectively oracle-like and usually resolves to a single malicious file read. The environment is deliberately small: one text-only local-files agent, no external side effects, and deterministic scoring. The broad artifact is stronger than the live section, and the live headline is retrieval-first because that is where the safety-retention-cost wedge is stable. Stealth persistence is weak in the current artifact, and rolling summaries remain more rollback-like than writer-repair-like. Finally, all recovery guarantees depend on sufficiently complete object-level logging; partial provenance would weaken or break the repair claim.

These limits are the price of a cleaner statement. Rather than broadening into a weaker benchmark paper, we present the narrowest result that the implementation strongly supports.

7 Conclusion

Selective revocation plus writer-only replay is not a general solution to prompt injection. It is a post-compromise recovery primitive for explicit persisted state. In the strongest supported setting—explicit attacks on retrieval memory—it removes residual attack influence more surgically than delete/reset baselines and more cheaply than rollback. The focused live confirmation shows that the same wedge survives in a real local model. Once persistent compromise has been written into tracked state, repair becomes a systems problem; this paper shows one narrow case where that repair can be done well.

Ethical Considerations

This project studies post-compromise recovery for a narrow prompt-injection setting using synthetic local workspaces and rule-based evaluation. No human subjects are involved, no live external systems are attacked, and no sensitive operational data is used. The attack files and restricted documents are intentionally small, synthetic, and local to the artifact. The goal is to measure how persistent state can influence later behavior and how targeted recovery can remove that influence. We do not release any capability for broader compromise beyond the described local testbed. The paper's live-model section uses an offline local model under a schema-constrained interface rather than a public deployment, which further limits external risk.

Open Science

The artifact accompanying this paper contains the full implementation, frozen task catalog, run scripts, raw result files, generated tables and figures, and a paper package. The artifact supports deterministic reproduction of the full matrix and focused reproduction of the live confirmation batch. In particular, the artifact includes commands to rebuild the raw matrix results, regenerate all reported paper tables and figures, and compile this LaTeX paper source. The standalone paper folder is self-contained: its tables, bibliography, and figure assets are bundled locally rather than loaded from external result files at compile time. The live confirmation section is intentionally focused and includes the exact repeated retrieval settings used for the main claim. The artifact also records exploratory neg-

ative pilots, including unstable summary-memory live runs and a 32B local-model run that did not improve the tradeoff.

References

[1] Zhaorun Chen, Zhen Xiang, Chaowei Xiao, Dawn Song, and Bo Li. Agentpoison: Red-teaming llm agents via poisoning memory or knowledge bases. In *Advances in Neural Information Processing Systems*, 2024. NeurIPS 2024 poster.

[2] Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu, and Zhen Xiang. Memory injection attacks on llm agents via query-only interaction. In *Advances in Neural Information Processing Systems*, 2025. NeurIPS 2025 poster.

[3] OWASP GenAI Security Project. Llm01:2025 prompt injection. https://genai.owasp.org/llmrisk/llm01-prompt-injection/, 2025.

[4] Saksham Sahai Srivastava and Haoyu He. Memorygraft: Persistent compromise of llm agents via poisoned experience retrieval. *arXiv preprint arXiv:2512.16962*, 2025.

[5] UK National Cyber Security Centre. Prompt injection is not sql injection (it may be worse). https://www.ncsc.gov.uk/blog-post/prompt-injection-is-not-sql-injection, 2025.

[6] Ruoyao Wen, Hao Li, Chaowei Xiao, and Ning Zhang. Agentsys: Secure and dynamic llm agents through explicit hierarchical memory management. *arXiv preprint arXiv:2602.07398*, 2026.

[7] 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*, 2026.

[8] Tian Zhang, Yiwei Xu, Juan Wang, Keyan Guo, Xiaoyang Xu, Bowen Xiao, Quanlong Guan, Jinlin Fan, Jiawei Liu, Zhiquan Liu, and Hongxin Hu. Agentsentry: Mitigating indirect prompt injection in llm agents via temporal causal diagnostics and context purification. *arXiv preprint arXiv:2602.22724*, 2026.

[9] Qiming Zhu et al. From lossy to verified: A provenance-aware tiered memory for agents. *arXiv preprint arXiv:2602.17913*, 2026.