OWASP Ranks Memory Poisoning ASI06, and Detectors Miss 66% of Poisoned Entries

A user chats normally with your agent. No database access, no credentials. Weeks later the agent acts on a memory that user planted, and the reviewer model that read it called it fine.

OWASP's Top 10 for Agentic Applications, published 9 December 2025, ranks Memory and Context Poisoning as ASI06, and on 13 May 2026 OWASP followed up with a post calling it a live attack surface rather than theory. The worse number comes from elsewhere: A-MemGuard reports that advanced LLM-based detectors miss 66% of poisoned memory entries, because each one looks benign reviewed on its own.

ASI06 covers persistent corruption across conversation history, RAG stores, vector databases, knowledge graphs and long-term memory. That scope is the point. Prompt injection is a single-turn problem you can fail closed on. Memory poisoning is a write to durable state, and the write outlives the session that made it.

The timing is what makes this worth reading now rather than in 2027. OpenAI announced "Dreaming: Better memory for a more helpful ChatGPT" on 4 June 2026, initially for Plus and Pro users in the US. Claude memory is on by default for Free, Pro and Max, off by default for Team and Enterprise, with sensitive topics like health and beliefs excluded unless explicitly enabled. Persistent profiles are now the default consumer experience.

66%
poisoned memory entries missed by advanced LLM detectors
A-MemGuard, arXiv:2510.02373, Oct 2025

98.2%
MINJA average injection success on GPT-4 and GPT-4o agents
arXiv:2503.03704, Mar 2025

76.8%
MINJA average attack success on the same agents
arXiv:2503.03704, Mar 2025

Why an LLM reviewer misses two thirds of poisoned entries

Because it reads one entry at a time, and one entry at a time is the wrong unit of analysis. A poisoned memory is usually a chain: several plausible statements that only produce a harmful action once they are retrieved together. A-MemGuard's framing is that malicious content appears benign in isolation, which is exactly the condition that defeats a per-item classifier. You are asking a model to judge a sentence whose meaning lives in its relationship to other sentences it cannot see.

MINJA (arXiv:2503.03704, submitted 5 March 2025) shows how cheap the write is. The attack works through query-only interaction: a normal user talking to the agent, as The Register reported on 11 March 2025, with no backend or database access required. Across RAP, EHRAgent and a QA agent on GPT-4 and GPT-4o, Shen Dong, Shaochen Xu, Pengfei He, Yige Li, Jiliang Tang, Tianming Liu, Hui Liu and Zhen Xiang report 98.2% average injection success and 76.8% average attack success.

The clinical numbers are the ones I keep coming back to. EHRAgent reached 90.0% attack success on eICU and 57.0% on MIMIC-III. Same attack, same agent, two clinical datasets, a 33 point spread. Nothing in the published data explains that gap, and I am not going to invent a reason. Read it as evidence that your own exposure is dataset-specific and cannot be inferred from someone else's benchmark.

! **The catch with defenses**
A-MemGuard reports cutting attack success rate by over 95% across benchmarks, taking EHRAgent from 100.0 to as low as 2.13 at retrieval. That is a research result on research benchmarks, not a product you can install this quarter. Treat it as proof the problem is tractable rather than as a shipping mitigation.

The write path is wider than chat

Two 2026 papers extend this past the conversational case. arXiv:2606.04329 (3 June 2026) describes an experience-to-procedure write: the agent synthesizes a poisoned trace into a procedural skill, and the self-improvement loop reinforces it across sessions. The agent has not remembered a bad fact. It has learned a bad method, and every successful run of that method strengthens it.

A 2026 threat taxonomy (arXiv:2604.02837v1, 3 April 2026) documents memory file poisoning, where a skill instructs the agent to write adversarial content into AGENTS.md, MEMORY.md or SOUL.md. Those are files in your repository. They go through your review process, or they do not, and mostly they do not, because nobody treats a markdown context file as executable.

Palo Alto's Unit 42 showed the exfiltration variant on 9 October 2025: indirect prompt injection persisting in long-term memory, with the assistant sending user conversation history to an attacker-controlled domain. Persistence plus an egress path is the full chain.

WHERE I LAND

Most coverage of ASI06 treats memory as a new flavour of prompt injection and reaches for the same fix: put a classifier in front of it. The 66% miss rate says that fix is structurally wrong rather than badly tuned. My take: the useful mental model is an append-only database with no schema, no write authorization and no audit log, which your agent treats as ground truth. Frame it that way and the controls become the boring ones you already know how to build, with content inspection the least important of them.

Controls that do not depend on reading the entry

The defenses worth funding are the ones that work without correctly classifying content, because classification is the part that demonstrably fails.

Scope memory writes by origin. A memory produced during a session that touched untrusted external content (a fetched page, a tool result, an email) should be tagged as such and kept out of retrieval in privileged contexts. That is provenance, not sentiment analysis.

Log every write with its originating session, and make the log queryable. If MINJA-style injection lands at 98.2%, you will not prevent every write. You need to answer "what did the agent learn in March, and from whom" after the fact. **I would rather have a complete write log and no detector than a detector that catches a third of what it sees.**

Put memory files under code review. AGENTS.md, MEMORY.md and SOUL.md should need the same approval as a dependency bump. If an agent can write to them unattended, that is a privilege escalation path with a markdown extension.

Evaluate retrieval sets, not single entries. A-MemGuard's result comes from reasoning over related entries. Even a crude version, flagging when a retrieved set contains entries written in the same session that together change a tool-call decision, catches the chain structure per-item review misses.

Turn memory off where the blast radius is large. Claude already ships with memory off by default for Team and Enterprise. That default exists for a reason, and it is worth keeping until the write log exists. The same reasoning applies to tool surfaces generally, which is why I keep returning to how much of the MCP ecosystem ships without auth.

What the next twelve months probably look like

I expect the first publicly disclosed memory-poisoning incident against a production enterprise agent within twelve months, and I expect it to surface through odd agent behavior rather than through a detector, since detectors miss two thirds of entries. I also expect vendors to answer with a memory review UI for end users rather than a write-

authorization model, because the former ships faster and demos better.

This next part is my reading rather than anything in the research: the procedural-write path from arXiv:2606.04329 will be the hardest to clean up, because rolling back a corrupted skill means identifying every downstream action it produced. A poisoned fact you can delete. A poisoned method has already changed the record.

If you are running agents with persistent memory and cannot currently say where a given memory came from, that gap is the thing to close first, and it is a conversation I am happy to have before you spend budget on a classifier that will miss most of what you point it at.

WRITTEN BY

Artur Markus

Software engineer in Basel building AI infrastructure and agentic systems, with a background in pharmaceutical automation, GxP compliance, and validation. I help teams turn AI from a chatbot into a reliable operational layer.

Book a meeting →