What is RAG poisoning?

RAG stands for retrieval-augmented generation, the standard way modern AI products answer questions from a company's own documents. RAG poisoning is the attack where someone plants malicious content in the corpus the AI reads from (a user-uploaded PDF, a scraped web page, a shared wiki). Two failure modes: the planted content instructs the AI to do something wrong, or it feeds the AI false facts the AI then confidently repeats.

HOW IT WORKS

01 How does attacker content end up in a production RAG corpus?

More easily than most teams expect. The vectors we see most often on engagements:

- User-uploaded documents. Help-desk attachments, support-ticket bodies, customer-submitted PDFs. Any user input that ends up in the corpus.
- Auto-ingested third-party feeds. RSS, partner APIs, scraped web content, public ticket boards.
- Public web crawling. If the retriever fetches from the public web on the fly, every site the user mentions is a potential injection source.
- Internal-but-untrusted sources. A shared wiki where any employee can edit, a Slack export, a code-comment harvest. The boundary you thought you had inside the org is often porous.
- Embedding-space manipulation. Carefully crafted chunks designed to rank high for specific queries, sometimes through token-level adversarial methods against the embedding model.

02 How does SecureLayer7 test for RAG poisoning?

We enumerate every ingestion path into the corpus, classify each path's attacker reachability, and plant adversarial chunks of varying subtlety. We measure retrieval rank (do our chunks come back?) and impact (does the model honor the planted instruction, or assert the planted fact?). For systems with both retrieval and tool access, we chain the poisoned retrieval into a downstream action and prove blast radius. Every confirmed finding ships with the planted chunk,

HOW TO DEFEND

- Ingestion-time provenance. Tag every chunk with its source, ingestion path, and trust level. Surface that provenance in the prompt so downstream defenses can reason about it.
- Source-aware retrieval. Restrict which sources are eligible for which queries. A consumer-facing assistant should probably not retrieve from auto-ingested third-party feeds without an authority check.
- Adversarial corpus review. Run scheduled scans for known payload patterns, anomalous embeddings, and chunks with suspicious instruction-language density.
- Output verification against retrieved sources. When the model cites a fact, programmatically check that the cited chunk actually supports the claim. Catches a large fraction of knowledge-corruption attacks.
- Least-privilege downstream actions. If the RAG system feeds an agent that can take actions, the action layer should not trust the retrieval result transitively.

SOURCES

[1] OWASP LLM06:2025, Sensitive Information Disclosure

[2] OWASP LLM08:2025, Vector and Embedding Weaknesses

[3] Greshake et al., More Than You've Asked For

the query that retrieved it, the resulting model output, and an architectural recommendation.

[4] Zou et al., PoisonedRAG

Audit your RAG corpus before someone else does.

securelayer7.net/learn/ai-security/rag-poisoning

Open online