AI PRODUCT SECURITY IN THE AGE OF MYTHOS · 2026

Chapter 11 · RAG and Context Systems Are Data Security Systems

Standalone reading module for LMS delivery and required reading.

FORMAT	USE	SCOPE	AUDIENCE
Standalone PDF	Required reading	Single chapter	Learners

CHAPTER 11

RAG and Context Systems Are Data Security Systems

LLM08

VECTOR AND EMBEDDING WEAKNESSES

OWASP's 2025 LLM Top 10 calls out vector and embedding weaknesses as a distinct risk category for retrieval systems and context pipelines.

OWASP GENAI SECURITY PROJECT, 2025 LLM TOP 10

RAG authorization is not an AI problem. It is a data security problem.

AI security depends on data trust. The model is only the visible surface. The real control plane spans data classification, retrieval authorization, metadata quality, lineage tracking, identity boundaries, and evidence logging. An organization that cannot prove what data a model was allowed to see—or what data it actually received—cannot prove it governed its AI systems.

Similarity search is not permission checking.

The previous chapter explained how workflow chains amplify trust decisions: each step in an agentic workflow reads context, makes a decision, and triggers the next step. RAG systems are one of the most critical of those decisions because they decide what information the model is allowed to know before it acts. If retrieval happens before authorization, a workflow can leak secrets, corrupt reasoning, or violate tenant boundaries through chains of decisions that look individually reasonable.

RAG systems become security boundaries because they decide what information enters model context. If retrieval happens before authorization, the vector database can become an access-control bypass. The model may receive content the user should not see. Output filtering becomes a late, fragile control.

Search systems have always been security boundaries. Enterprise search, email discovery, file indexing, SIEM search, and analytics platforms all faced the same basic question: is the user allowed to see this result? RAG makes the problem sharper because the result may never appear as a direct quote. A private chunk can enter context, shape the answer, and disappear from the final output. The user sees a fluent answer. The incident reviewer has to prove what the model saw.

OWASP's LLM08 on vector and embedding weaknesses explicitly covers this: weaknesses in how vectors and embeddings are generated, stored, or retrieved can be exploited to inject harmful content, manipulate outputs, or access sensitive information. Microsoft's red-team lessons on more than 100 AI products identified cross-prompt injection attacks against RAG systems as a specific attack vector—hostile text in one document influencing retrieval and output of other documents.

The Permission Decay Problem

A realistic RAG failure often begins with a permission change that the index never learns about.

A product team builds a support assistant that answers questions about account health, billing, and deployment. They index documents from multiple sources: public help articles, internal runbooks, customer-account notes, and billing records. The vector index is built weekly. At ingestion time, a particular chunk—"Customer escalation notes for account XYZ"—is public within the organization because the support team needs to reference it.

Three weeks later, the account notes are reclassified. A customer complaint surfaces an issue that requires legal review. The document is now marked confidential, visible only to the legal and executive teams. The source system enforces this restriction. But the vector index was not rebuilt. The chunk is still indexed with the old permission level.

A support agent from a different team asks the assistant: "What are the key issues affecting account XYZ?" Similarity search finds the old

chunk. The vector database does not know the permissions have changed. The model receives the confidential escalation notes in its context. The answer does not quote the confidential sentences directly, but it summarizes enough to reveal that there is an ongoing legal issue. The support agent now knows something they should not know.

The source system was correct. The vector index was stale. The model did not misbehave. The authorization boundary failed before the model wrote the answer.

This is why output filtering is too late. Output filtering asks the question after the boundary has been crossed. By then, the private information is already in the model's context. The model might summarize it, might derive conclusions from it, might incorporate it into reasoning. Output filtering cannot unsee what the model has already processed.

> ***Similarity search is optimization, not authorization:*** *It finds relevant content fast, but does not check whether the user is allowed to see it. Authorization must happen before retrieval, not after.*

Multi-Tenant Contamination

A second failure pattern appears in shared retrieval systems.

A support platform offers AI-powered customer support across 200 different customer accounts. Each customer's account has tickets, help articles, runbooks, and internal notes. The platform embeds all of this content into a single shared vector index because maintaining 200 separate indexes is expensive and harder to tune. The index is updated daily with new chunks from all tenants.

The retrieval system works like this:

1. Customer X asks a question
2. The system embeds the question
3. Similarity search finds the 10 most-similar chunks across all tenants
4. The model reads those chunks and answers

The retrieval system knows which customer is asking (Customer X), but it does not filter the candidate set before similarity search. Similarity search is semantic, not authorization-aware. It returns the closest chunks regardless of tenant or permission.

Now suppose chunk 17 is from Customer Y's confidential escalation notes. And chunk 17 is semantically similar to Customer X's question—same product, same failure mode, but different context. The retriever selects chunk 17 because it is the most relevant. The model receives Customer Y's private information. Customer X's support agent now knows about Customer Y's issues.

The product expects the retriever to rank and the model to answer with discretion. But the model has already read Customer Y's information. Discretion cannot unsee what has been read.

The safe answer is not to depend on model restraint. The safe answer is retrieval eligibility. Before context assembly, the system filters the candidate set by: Is this chunk from the requesting customer's tenant? Does the requesting user have permission to see this classification? Has the source document been deleted? Are there any time-based restrictions?

> ***Cross-tenant leakage is a design problem, not a behavior problem:*** *It is fixed through pre-retrieval authorization checks, not through model tuning or output filtering.*

The Authorization-First Pattern

Permissions rarely survive the embedding pipeline by accident. Source systems have roles, groups, sharing links, document labels, expiry rules, deletion flows, and audit trails. The embedding pipeline may strip those details. The vector index may store chunks without chunk-level ACLs. The retrieval service may know the user but not check eligibility.

The safe pattern is authorization before retrieval, or at minimum before context assembly.

A proper retrieval system works like this:

1. User asks a question
2. System embeds the question
3. Similarity search returns candidate chunks
4. Before context assembly, the system checks: Is this chunk eligible for this user, in this tenant, with this role, at this time?
5. Only eligible chunks enter context
6. The model answers based on authorized information

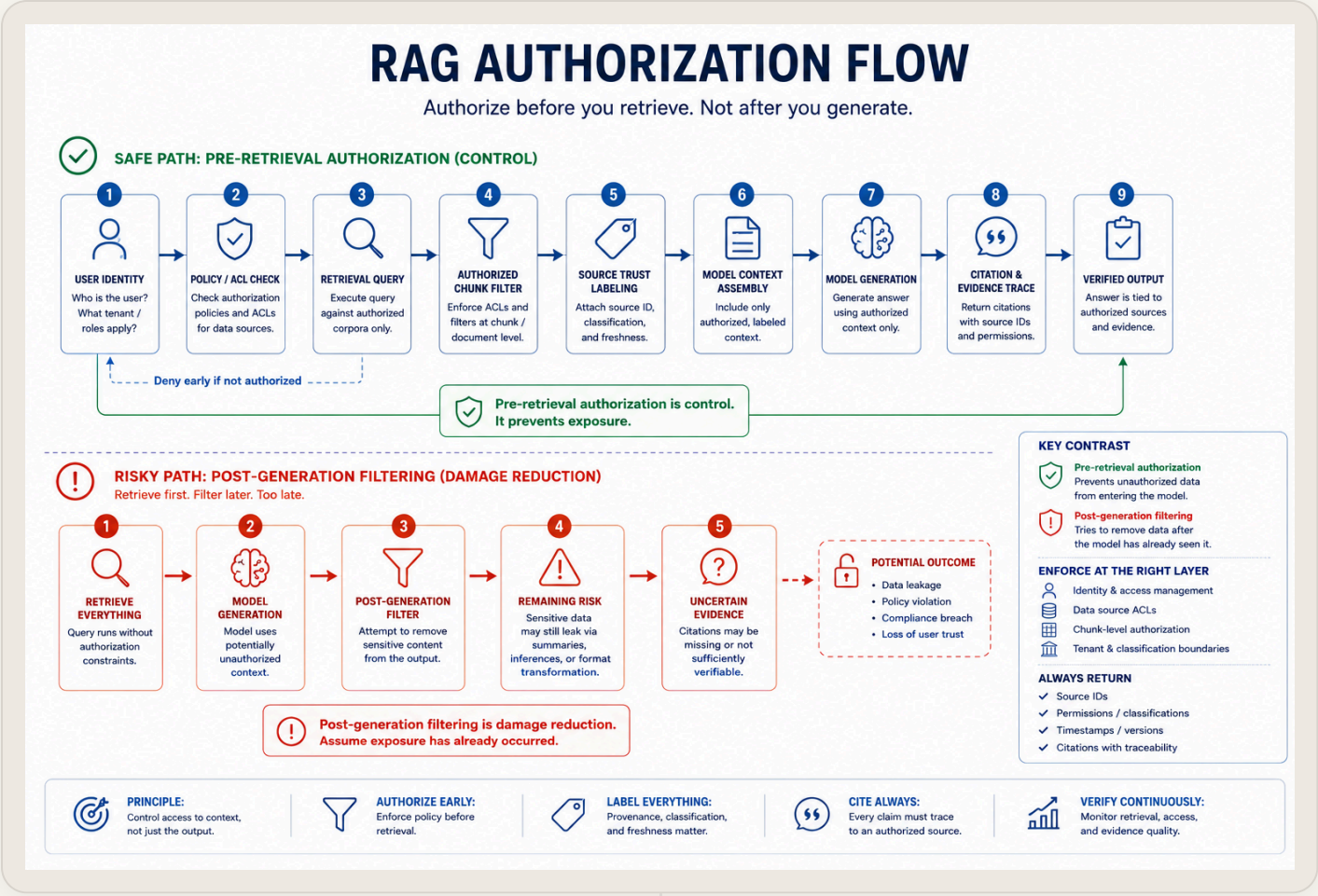

FIGURE 1: RAG AUTHORIZATION FLOWS FROM IDENTITY AND ACL CHECKS BEFORE RETRIEVAL, CONTRASTED WITH A DANGEROUS POST-GENERATION FILTERING PATH THAT CANNOT UNSEE UNAUTHORIZED DATA ALREADY IN THE MODEL'S CONTEXT.

The check at step 4 is not a suggestion. It is mandatory. The check requires knowing:

- **Source of truth for permissions** – Where does the system learn about ACL changes? The document's source system? A central policy store? These must stay in sync.
- **Sync frequency** – How often are permission changes propagated to the retrieval system? If the answer is "weekly," stale permissions can hide for up to 7 days.
- **Maximum tolerated staleness** – For a public knowledge base, staleness of weeks is acceptable. For customer data, staleness of hours is dangerous.
- **Deletion handling** – If a document is deleted in the source, is it removed from the index immediately, or does it linger?
- **Failure behavior** – If the system cannot verify permission, what happens? Fail closed (do not include the chunk) or fail open (include it)?
- **Audit trail** – Every retrieval eligibility decision should be logged. If there is a breach, the organization needs to know what private chunks were served.

For high-risk systems (multi-tenant, high-value data), additional controls may be necessary: tenant-isolated indexes, dedicated ACL stores, source-of-truth synchronization, chunk-level permissions, and retrieval audit logs.

For the support assistant, this means a customer question can retrieve only the customer's eligible tickets and approved knowledge-base material. Internal escalation notes, other tenants' tickets, deleted articles, and billing-policy drafts must fail eligibility before they become model context.

The Uncomfortable Tradeoff

The best semantic match may not be an eligible chunk. The most useful document may belong to another tenant. The freshest content may have a confidentiality boundary. The highest-ranking result may be legally sensitive.

Search optimization is a different goal than authorization. The retriever must prefer the best authorized answer over the best answer. If that means serving a slightly less-relevant chunk that the user is authorized to see, that is the correct tradeoff. If it means returning no answer because all the best matches are unauthorized, that is also correct.

RAG security is not a search-quality problem. It is authorization design.

Evaluating RAG Authorization

Before claiming a RAG system is secure, test these scenarios:

TEST IDENTITY	QUERY	EXPECTED RETRIEVAL	FORBIDDEN RETRIEVAL	EVIDENCE REQUIRED
Support agent, Customer A	"customer account status"	Docs from Customer A public folder	Customer B's account notes, any confidential docs	Retrieval trace showing eligibility check
Manager, Department Finance	"Q3 budget"	Q3 budget doc they can access	Confidential personal salary data, other team budgets	ACL check log before context
Junior engineer	"authentication code review"	Public security guidance	Internal credentials, system secrets, pending security patches	Source-of-truth ACL, timestamp
Admin user (special token)	Same query as junior	All docs (including denied ones)	None (admins should see everything)	Permission verification against role
Deleted user account	"support request"	None (account should not retrieve)	Any docs (old sessions could be compromised)	Access check rejects deleted identity
User after role change	Same query as before + after role change	Only docs for new role	Docs from old role still in cache	ACL change timestamp vs. index rebuild

For each test, the system must show:

1. **Pre-retrieval authorization check** passed or failed
2. **Timestamp** of permission state at retrieval time
3. **ACL source of truth** that was consulted
4. **Chunks included** in context (so incident response can audit what was seen)
5. **Chunks excluded** (so you can see which dangerous data was correctly blocked)

Stale Permission Recovery

Permission staleness is inevitable. The question is how the system handles it.

Best: Document ACLs are versioned, indexed with chunk metadata, and checked pre-retrieval. If ACLs are stale (> max tolerated age), retrieval

fails closed.

Acceptable: Permissions are checked at ingestion time and stored with chunks. If a document is reclassified, the index is re-ingested within SLA (e.g., 2 hours for customer data, 1 day for public docs).

Dangerous: Permissions are checked post-retrieval (output filtering). By then, the model has already seen unauthorized data.

Very Dangerous: The index knows the source but does not verify permissions at retrieval time. Threat: stale permissions, deleted documents, tenant boundaries.

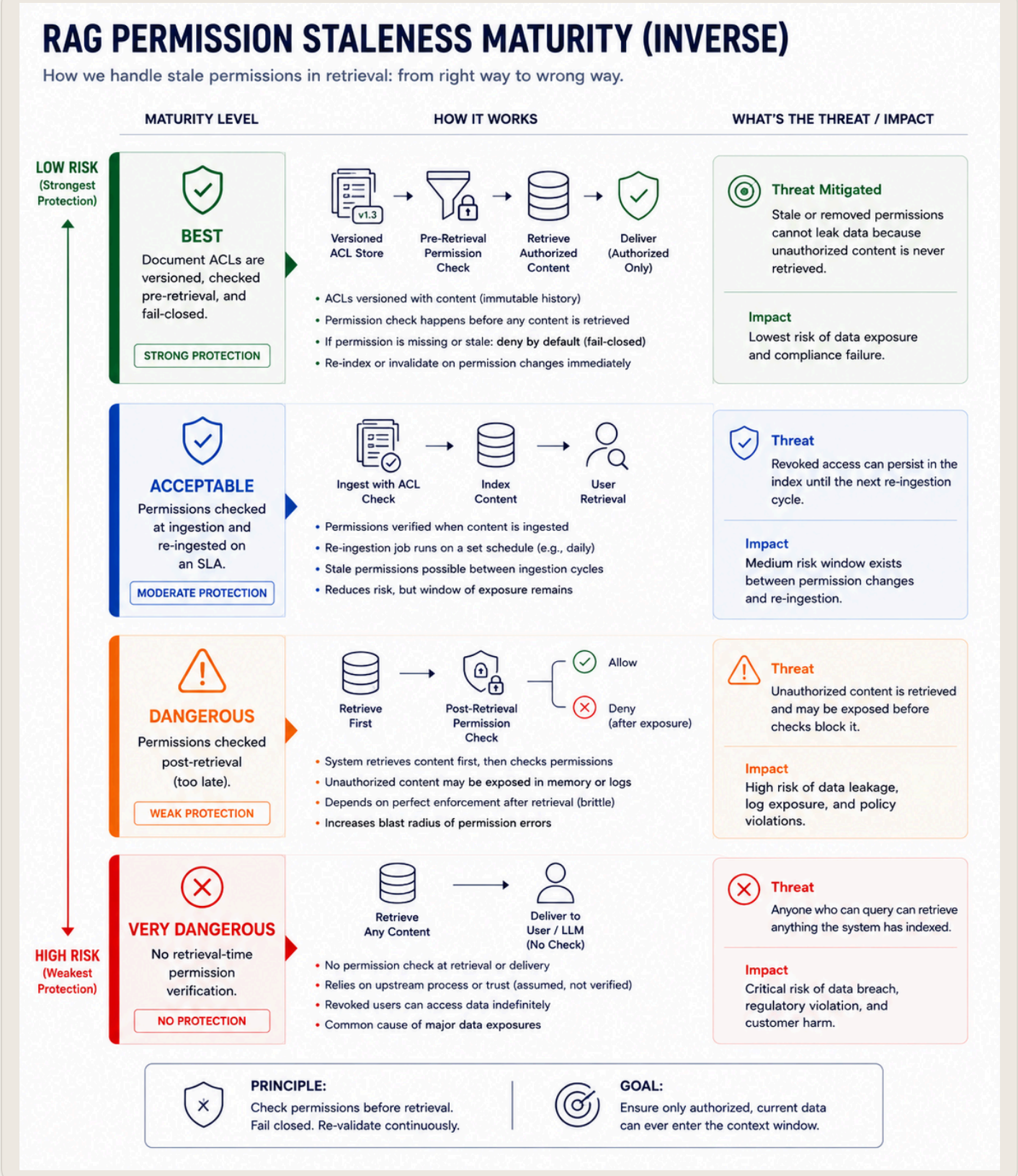

FIGURE 2: RAG PERMISSION STALENESS HANDLING: FROM BEST (VERSIONED ACLS, PRE-RETRIEVAL CHECKS, FAIL-CLOSED) THROUGH ACCEPTABLE (INGESTION-TIME CHECKS WITH SLA RE-INGESTION) TO DANGEROUS (POST-RETRIEVAL FILTERING) AND VERY DANGEROUS (NO RETRIEVAL-TIME VERIFICATION)—THE MATURITY LADDER INVERTED TO SHOW HOW TO BREAK RAG AUTHORIZATION.

Detailed RAG failure patterns, authorization review templates, ACL sync strategies, multi-tenant design patterns, and evidence collection procedures – in Appendix F.

Sources

- OWASP Top 10 for LLM Applications 2025: https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025

- Microsoft, Lessons from Red Teaming 100 Generative AI Products: https://openreview.net/pdf?id=auiAIKsJXg