AI SECURITY · FIELD NOTES

Prompt Injection & LLM Security: What Actually Breaks

> *“A system prompt is a suggestion the model usually follows, not a security boundary. Treating it as one is the single most common LLM-security mistake.”*

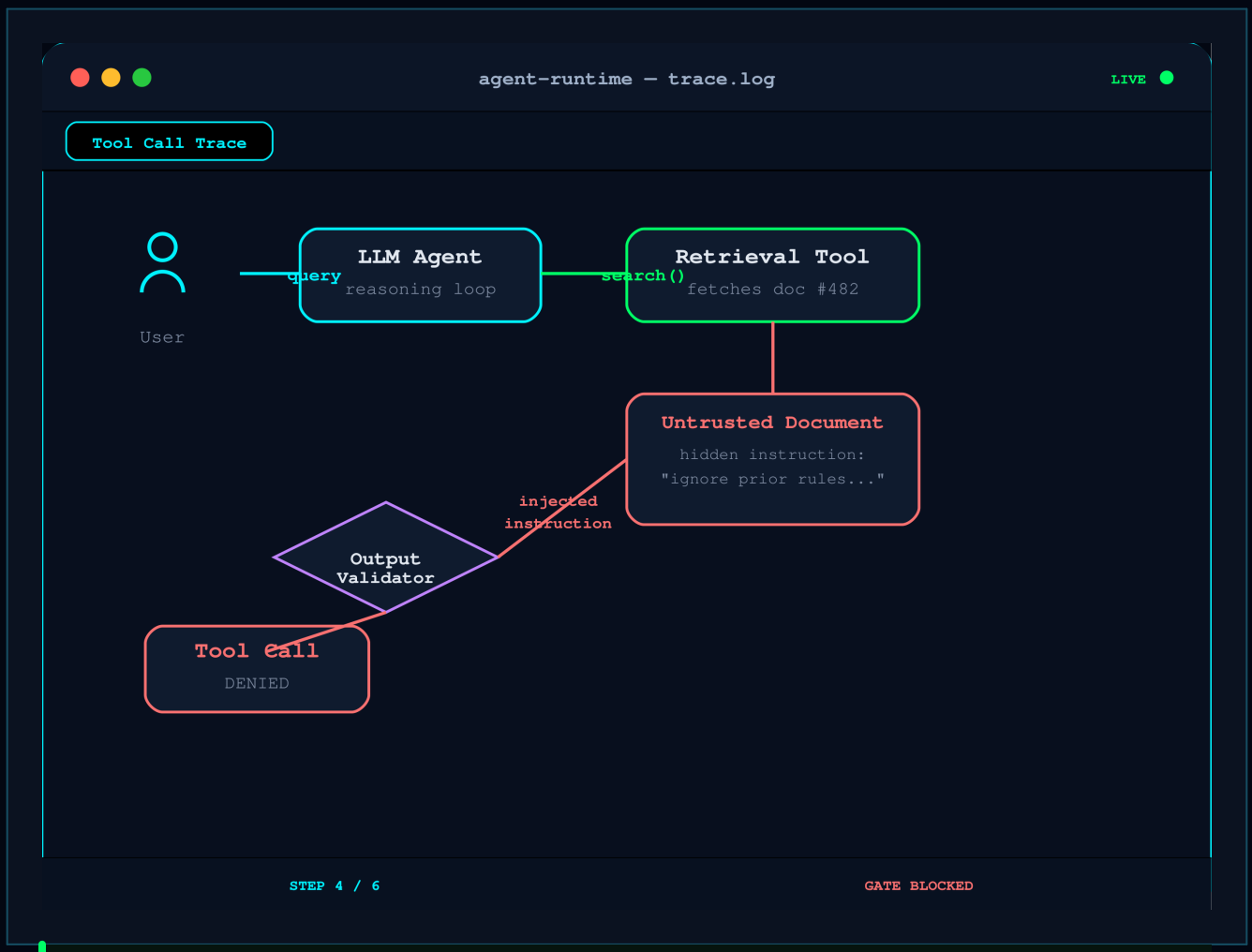
As AI agents move from chat demos into production — reading email, browsing retrieved documents, calling tools with real side effects — prompt injection stops being a novelty and becomes an actual attack surface with real consequences.

Scope	LLM Applications, RAG Pipelines & AI Agents
Threat Classes	Direct & Indirect Prompt Injection
Focus	Practical mitigations, not theoretical defenses
Analyst	Nazline Mwita, Cybersecurity Assurance Lead, HarLyn Digital Partners

1. Direct vs. Indirect Injection

Direct injection is a user typing an instruction meant to override the system prompt (“ignore previous instructions”). It's the obvious case, and most teams at least attempt to filter for it.

Indirect injection is more dangerous and far more common in production RAG/agent systems: the malicious instruction doesn't come from the user at all. It's embedded in a document, email, web page, or API response the agent retrieves and treats as data — except the model doesn't reliably distinguish “data to summarize” from “instructions to follow.”



Security Principle

Anything the model reads that you did not author is untrusted input, including retrieved documents and tool output.

2. Why Naive Defenses Fail

“Add a stronger system prompt telling it not to follow injected instructions” is the most common first response, and it does not reliably work. The model has no hard boundary between instruction-tokens and data-tokens at the architecture level — a sufficiently crafted payload in retrieved content can still shift model behavior regardless of system-prompt wording.

3. What Actually Helps

Control	What it does
Tool permission boundaries	Bound what the agent can DO regardless of what it's told
Human approval gates	Require confirmation before consequential/irreversible actions
Output validation	Check tool-call arguments against a schema before execution
Retrieval provenance	Track and surface where retrieved content came from
Sandboxed execution	Isolate tool execution from credentials/systems it doesn't need
Trajectory logging	Record what the agent actually did, not just what it said

4. The Core Shift in Thinking

The realistic security posture is not “prevent all injection” — that's not currently achievable with general-purpose LLMs. It's “assume injection will sometimes succeed, and bound the blast radius of what a successfully-injected agent can actually do.”

- Every tool call an agent can make should have the minimum permission needed, not broad API access.
- Consequential actions (sending money, deleting data, sending external communications) need a human approval gate, not just a confidence threshold.
- Retrieved content should be visibly tagged as untrusted context to the model, and validated on the way out.

Related Services

This dossier supports the RAG Security and AI Agent Security service pages — technical reviews of retrieval pipelines, tool-call boundaries, and agent permission architecture.

Nazline Mwita, CompTIA Security+ certified Cybersecurity Assurance Lead and Co-Founder at HarLyn Digital Partners. Specializing in authorized web & API security assessments, KDPA compliance reviews, and defensive cloud architecture in Nairobi, Kenya.