Xtensyon.

WHITE PAPER

# Prompt Injection in Enterprise Systems: Threats, Tests, and Practical Defenses

July 3, 2025 • Xtensyon Labs • 10 min read

> *Prompt injection is not a theory problem. It shows up through emails, PDFs, tickets, and chat logs. This paper lays out a hardening checklist that security teams can validate and engineers can ship.*

## TL;DR

- Treat retrieved content as untrusted input, even if it comes from internal systems.
- Separate system prompts, tool rules, and user content with strict boundaries.
- Add automated tests that include malicious documents and tricky user requests.
- Log tool calls and retrieval context so incidents are traceable.

## Executive Summary

Enterprise LLM apps ingest documents written by many people for many reasons. Attackers do not need special access. They only need content that the model will read. This paper provides a practical view of prompt injection in RAG and tool-using agents, then maps controls to real engineering work: content sanitization, policy enforcement, sandboxed tools, and automated regression tests.

## Why It Matters

If a model can be tricked into changing rules, leaking data, or calling tools incorrectly, you will have incidents. The risk is higher in enterprise settings because inputs include contracts, emails, and customer tickets. A security plan that stays at the slide level will not help. What works is a mix of guardrails, tests, and logging.

## What We Built

- A content ingestion step that strips hidden text, removes dangerous links, and flags suspicious patterns.
- A policy layer that blocks tool calls outside allowed scopes and validates tool arguments.
- A red-team test suite: malicious docs, indirect prompts, and data exfil scenarios.
- Tracing for retrieval and tool calls, including allow or block decisions.

## Observed Outcomes

- Fewer unsafe tool calls after adding argument validation and scope checks.
- Faster incident triage because traces showed what the model saw and what it tried to do.
- More confidence during releases after adding injection cases to regression tests.

## Implementation Notes

- Do not rely on “please ignore” wording in prompts. Treat prompts as policy signals, not locks.
- Keep tool permissions small and explicit. Add timeouts and rate limits.
- Use allowlists for URLs and domains before fetching external content.
- When in doubt, refuse and ask for human confirmation.

## Governance & Risk

- Decide what logs are needed for audits and incidents, then implement retention rules.
- Train teams to write safe documents. Security is also a content problem.
- Review third-party plugins and tools as part of app security.

## Release Checklist

- Are retrieved documents treated as untrusted input?
- Are tool calls validated and scoped?
- Do we run injection regression tests on every release?

- Do logs capture retrieval, prompts, and tool decisions?
- Is there a safe refusal path for risky requests?

## Conclusion

Prompt injection defense is engineering work. When controls are built into ingestion, tooling, and testing, the risk becomes manageable and measurable.

## Keywords

prompt injection | security | llm appsec | rag | threat modeling | policy