# What is indirect prompt injection?

Indirect prompt injection is the version of prompt injection where the attacker plants instructions in content the AI will read on someone else's behalf, like a web page the AI summarizes, a support ticket it reads, or a file it ingests. The user never sees the attack. This is the failure pattern that turns AI assistants from a user-experience risk into a real security risk.

## HOW IT WORKS

### 01 Which channels carry indirect prompt-injection payloads in production?

The list grows every quarter. The ones we test on every engagement:

- RAG-indexed documents. Anything the retrieval pipeline pulls into context. If users upload PDFs, scrape web content, or sync from third-party SaaS, every one of those is an instruction channel.
- Tool / function responses. A search tool that returns attacker-controlled snippets, an email tool that returns inbox bodies, a database tool that returns user-supplied notes.
- Markdown and HTML rendered to the model. Alt text, link labels, hidden Unicode, CSS-hidden spans, comments.
- Image content with OCR. A payload printed inside the image bytes that the OCR pipeline lifts out and feeds back to the model.
- JSON or YAML fields. A nested string the model is supposed to summarize, where the string itself is an instruction.
- Calendar invites, contact card notes, file names. Anywhere the agent processes structured data that originated outside your trust boundary.

### 02 What real-world indirect prompt-injection cases should I read?

- Greshake et al., 2023, the foundational paper. Demonstrated exfil through a markdown image link rendered by Bing Chat. Reading list day one.

## HOW TO DEFEND

- Provenance tagging of every chunk in the prompt (system vs operator vs user vs retrieved), explicitly marked so downstream defenses can reason about source.
- Render-boundary hardening: strip auto-resolving markdown links, sandbox image rendering, disallow inline scripts in model output that the UI honors.
- Least-privilege tool wiring so that even a fully compromised model cannot perform actions the user has not authorized.
- Second-model verification before any high-impact action (sending money, granting access, mutating production). The verifier sees only the action and a structured summary, not the original user input.
- Adversarial monitoring that alerts on tool-call shapes you have never seen, or on output that contains exfil-channel markers (long base64 strings inside URLs, for example).

## SOURCES

[1] Greshake et al., More Than You've Asked For (indirect prompt injection)

[2] OWASP LLM01:2025, Prompt Injection

[3] Cloak and Honey Trap: Defending LLM Agents

[4] MITRE ATLAS

- Cloak and Honey Trap (USENIX Security '25), Ben-Gurion researchers classified 7 LLM-agent vulnerability classes, 6 attacker strategies, and 15 attack techniques targeting agentic systems. The CHeaT testbed reproduces every one of them.
- Google Bard email leak (2024), indirect injection through a shared Google Doc caused the assistant to leak unrelated Gmail content.
- Bing Chat history exfil (2023), the canonical exfil-via-rendered-link case, still relevant as a template for every UI that turns model output into a network request.

## 03 How does SecureLayer7 test for indirect prompt injection?

We map every channel the model reads from before sending payloads: RAG corpus, tool response shapes, attached document parsers, OCR pipelines. We plant payloads in each channel and observe whether the model follows them. For agentic systems, the success criterion is not 'did the model say something it should not' but 'did the model perform an action it should not, or reach a resource outside its authorized scope.' Every confirmed finding ships with a reproducible transcript, the trust boundary that was crossed, and an architectural recommendation, not just a filter rule.

**Test your RAG and agentic pipelines for indirect injection.**
securelayer7.net/learn/ai-security/indirect-prompt-injection

Open online