CS-111 · AI Security

Prompt Injection

How attacker-controlled text hijacks the instructions of an AI system, and how to defend against it.

Executive Summary

Prompt injection is an attack in which text supplied by an attacker overrides or subverts the instructions a language model was meant to follow. Because these systems treat instructions and data as the same stream of words, a well-placed message inside user input or a fetched web page can redirect the model to leak data, misuse connected tools, or produce harmful output. It is widely regarded as the top security risk for applications built on large language models.

What It Is

Prompt injection is the manipulation of an AI system through crafted input so that it ignores its original guidance and follows the attacker instead. A language model does not have a hard boundary between the developer's instructions, the user's request, and any outside content it reads. All of it arrives as text, and the model tries to satisfy the most compelling instruction it sees. Direct prompt injection happens when a user types adversarial text straight into the system. Indirect prompt injection is more subtle and often more dangerous: the malicious instructions are hidden in content the model retrieves on its own, such as a web page, a document, an email, or a code comment, so the user never sees them and never consented to them.

Why It Matters

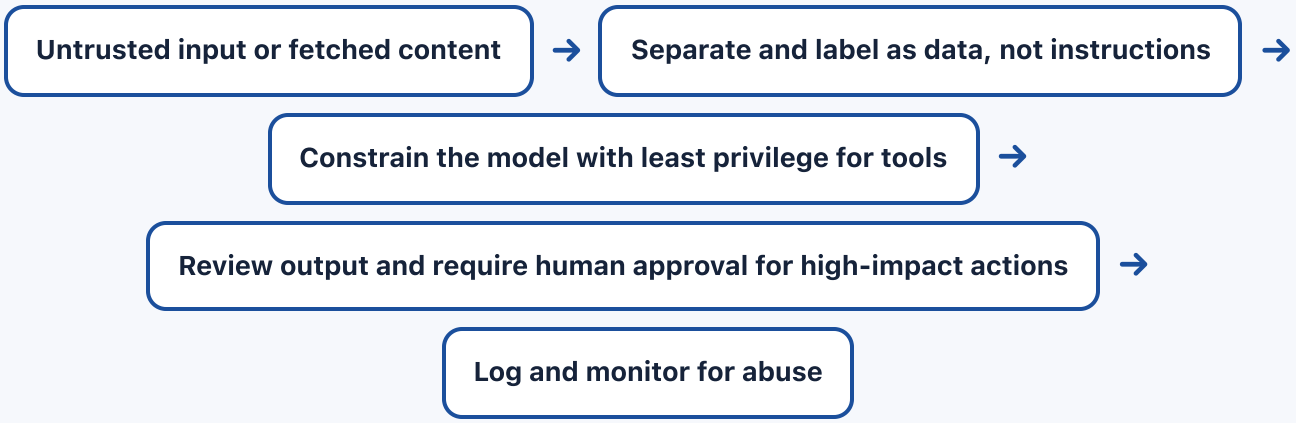
Organizations are wiring language models into email, customer support, code, databases, and internal knowledge, and they are giving those models tools that can send messages, run queries, and take real actions. A successful injection can turn a helpful assistant into a channel

for data theft, fraud, or sabotage without touching the underlying code. Because the attack rides on natural language rather than a software flaw, traditional defenses like input validation for known bad characters do not fully apply. For governance, risk, and compliance professionals, prompt injection is a fresh category of risk that maps to data protection, access control, and vendor oversight obligations, and it needs to be documented and managed like any other material threat.

How It Works

The core weakness is that the model blends trusted and untrusted text and then acts on the combined result. In a direct attack, the adversary crafts a message that persuades the model to disregard its rules, reveal its hidden instructions, or perform an off-limits action. In an indirect attack, the adversary plants instructions in content the model will later read, so the payload activates when an ordinary user asks the assistant to summarize a page, read an inbox, or process a file. The impact grows sharply when the model can use tools or plugins, because a hijacked model may be steered into calling those tools with attacker-chosen inputs. Defense therefore focuses on limiting what the model can reach, separating and labeling untrusted content, checking outputs before they are trusted, and keeping a human in the loop for consequential actions. This reference explains the risk and its mitigations only and does not provide working attack text.

Architecture Diagram



Untrusted content can carry hidden instructions; layered controls separate, constrain, and review it before any action is taken.

Visual Workflow

Map every place untrusted text can enter the system, including user input and any content the model retrieves.

→

Keep developer instructions, user input, and outside content in clearly separated and labeled channels.

→

Grant the model the least privilege it needs, and gate any tool that can act with explicit checks.

→

Validate and sanitize model output before it is displayed, executed, or passed to another system.

→

Require human approval for consequential actions such as sending money, deleting data, or emailing outside parties.

→

Log prompts, retrieved content, tool calls, and outputs, and monitor for signs of manipulation.

Common Attacks

- Direct injection where a user tells the assistant to ignore its rules or reveal its hidden instructions
- Indirect injection where hidden instructions sit inside a web page, document, or email the model reads
- Data exfiltration where the model is steered into leaking secrets, private records, or its own configuration

- Tool and plugin abuse where a hijacked model calls connected tools with attacker-chosen inputs
- Instruction smuggling that hides directives in formatting, metadata, or content the user never sees

Common Mistakes

- Assuming a strong system prompt alone can stop injection
- Trusting content the model fetches on its own as if a person had approved it
- Giving the model broad tool access and standing permissions it rarely needs
- Passing model output straight into code, queries, or downstream systems without checks
- Skipping logging, so an injection attempt leaves no evidence to investigate

Best Practices

- Treat all user input and all retrieved content as untrusted by default
- Separate and label instructions versus data so the model can weigh them differently
- Apply least privilege to tools and plugins, and scope every credential narrowly
- Validate output before it is trusted, and constrain it to expected formats where possible
- Keep a human in the loop for high-impact actions and irreversible operations
- Test against known injection patterns and add monitoring for anomalous behavior

Quick Checklist

- [x] Untrusted input and retrieved content are clearly separated from system instructions
- [x] Every connected tool runs with least privilege and narrowly scoped credentials
- [x] High-impact actions require human approval before they execute
- [x] Model output is validated and sanitized before display or downstream use
- [x] Prompts, tool calls, and outputs are logged and monitored
- [x] The application is tested against a current set of injection scenarios

Recommended Tools

LLM guardrail and filtering layer
Screens inputs and outputs for policy violations and manipulation attempts

Output validation and schema enforcement
Forces model responses into expected formats before they are trusted

AI gateway or proxy
Centralizes access control, rate limits, and logging for model traffic

Retrieval controls with content labeling
Marks fetched content as untrusted data rather than instructions

Industry Standards

OWASP Top 10 for LLM Applications
Names prompt injection as a leading risk for language model applications

NIST AI Risk Management Framework (AI RMF, AI 100-1)
Provides the Govern, Map, Measure, and Manage functions for handling AI risk

MITRE ATLAS
Catalogs adversarial techniques against AI systems for threat modeling

Career Relevance

Prompt injection is core knowledge for AI security engineers who harden model-powered applications, and for AI governance analysts and AI risk managers who must document and treat it as a named risk. GRC analysts increasingly assess it during vendor reviews, data protection assessments, and control testing for any product that embeds a language model. Fluency here signals that a candidate understands the security implications of generative AI, a fast-growing expectation for the roles AI-Governance-Jobs.com serves.

Interview Questions

- What is prompt injection, and how does indirect injection differ from direct injection?
- Why does a strong system prompt fail to fully prevent injection on its own?

- How does least privilege for tools and plugins reduce the impact of a successful injection?
- What role does human oversight play in defending high-impact AI actions?
- How would you test an AI application for prompt injection without publishing attack payloads?

Related Certifications

OWASP resources for LLM application security

ISC2 or ISACA AI security offerings (as available)

CompTIA Security+ (for the security foundations)

Further Reading

- OWASP Top 10 for LLM Applications
- NIST AI Risk Management Framework
- MITRE ATLAS

Key Takeaways

- Prompt injection works because models blend trusted instructions with untrusted text and act on the result.
- Indirect injection hides instructions in content the model retrieves, so users never see the attack.
- Impact scales with the tools and permissions the model holds, so least privilege is essential.
- Defense is layered: separate untrusted content, validate output, and keep humans in the loop.
- It is a named, material risk that AI governance, security, and GRC roles must manage.

FAQ

▶ **Is prompt injection the same as jailbreaking?**

▶ **Can I fully prevent prompt injection?**

▶ **Why is indirect prompt injection considered so dangerous?**

Related Careers

RELATED ROLES

- Ai Security Engineer
- Ai Governance Analyst
- Ai Risk Manager
- Grc Analyst

RELATED CERTIFICATIONS

- OWASP resources for LLM application security
- ISC2 or ISACA AI security offerings (as available)
- CompTIA Security+ (for the security foundations)

CURRENT OPENINGS

Live roles are on the job board.

Browse all jobs

GUIDES & SALARY

- Career guides
- Role roadmaps
- Salary data
- Career tools

SUGGESTED LEARNING PATH

1. Ground the basics with CS-001 Cybersecurity
2. Study this sheet: **Prompt Injection**
3. Go deeper: Model Poisoning
4. Go deeper: OWASP Top 10 for LLM Applications
5. Validate it: work toward **OWASP resources for LLM application security**
6. Find the role: browse current openings

RELATED SHEETS

CS-112 Model Poisoning | CS-116 OWASP Top 10 for LLM Applications | CS-115 Secure AI Adoption | CS-001 Cybersecurity | CS-064 Secrets Management