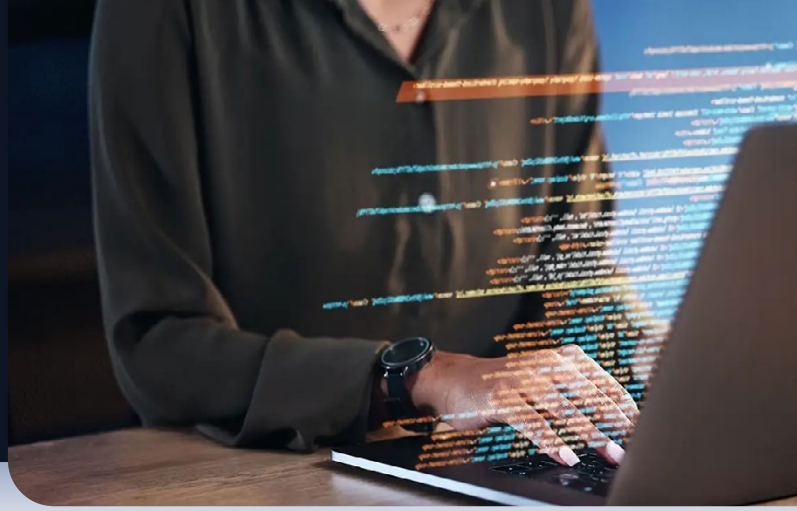
MINDGARD

How to Prevent Prompt Injection Attacks: Cheat Sheet



This cheat sheet translates prompt injection theory into practical controls teams can apply in real systems. Instead of treating prevention as absolute, it frames detection, mitigation, and protection based on deployment risk and system maturity.

Use it to quickly assess where your LLM workflows sit today, identify missing controls, and align defenses with the real-world impact of failure.

Prompt Injection Defense by Risk Level

Prompt injection defenses are not one-size-fits-all. The right controls depend on what the system can access, what actions it can take, and what failure would actually mean in production.

Low-Risk Systems

Examples Content generation | Internal experimentation | Marketing copy | Draft summarization

Required Detection

- Input logging with basic anomaly flags
- Simple keyword and pattern detection for override attempts
- Monitoring for repeated prompt manipulation attempts

Required Mitigation

- Constrained prompt templates
- Clear separation of system and user instructions
- Output filters for restricted content

Required Protection

- No tool access
- No sensitive data access
- No persistent memory

Failure Impact Output quality issues | Policy violations without real-world consequences

Medium-Risk Systems

Examples RAG over internal documents · Support assistants · Knowledge base search · Internal research tools

Required Detection

- Trust labeling for retrieved content
- Detection of instruction-like language inside documents
- Logging of retrieval context alongside prompts and outputs

Required Mitigation

- Sanitization of retrieved data before insertion into prompts
- Explicit constraints on how retrieved text may be used
- Output validation for sensitive data exposure

Required Protection

- Read-only access to internal data
- No action-taking tools
- Scoped context windows to limit instruction bleed

Failure Impact Data leakage · Incorrect or misleading responses · Compliance risk

High-Risk Systems

Examples Tool-calling agents · Financial actions · Infrastructure changes · Automated workflows with side effects

Required Detection

- Real-time monitoring of tool calls and parameters
- Behavioral anomaly detection across multi-step workflows
- Correlation between input source and action severity

Required Mitigation

- Fixed-role enforcement outside the model
- Strict allowlists for tool invocation
- Pre-execution policy checks that cannot be bypassed by prompts

Required Protection

- Least-privilege permissions for every tool
- Human approval for irreversible or high-impact actions
- Kill switches and automatic containment

Failure Impact Unauthorized transactions · Infrastructure compromise · Business-critical outages

Prompt Injection Defense Checklist for Production Teams

Use this checklist to validate whether your system is ready for real-world deployment.

System Design

- [ ] Attack surface fully mapped across inputs, tools, memory, and retrieval
- [ ] All inputs labeled by trust level
- [ ] Trust boundaries explicitly defined outside the model

Detection

- [ ] Inputs, outputs, tool calls, and retrieval context logged
- [ ] Alerts configured for anomalous behavior
- [ ] Indirect prompt injection monitored in RAG pipelines

Mitigation

- [ ] Output validation enforced for sensitive content
- [ ] Least privilege applied to every tool and API
- [ ] Prompt templates constrained and version-controlled

Protection

- [ ] High-risk actions gated by human review
- [ ] No implicit trust in retrieved or uploaded content
- [ ] Clear separation between instruction, data, and execution

Response

- [ ] Incident response playbook documented
- [ ] Containment steps defined for compromised workflows
- [ ] Rollback and access revocation procedures tested

Ongoing Assurance

- [ ] Continuous testing in place for prompt injection attempts
- [ ] Regular red-teaming against real workflows
- [ ] Defenses reviewed as capabilities and permissions evolve

Prompt injection prevention only works when defenses match system risk. Treat mitigation as contextual, enforce protection outside the model, and assume attackers will test every trust boundary you leave undefined.