Prompt Injection Defences — One-Page Checklist

Companion to Agentic AI Security Training — Chapter 4
Use this checklist when designing or reviewing any agent that processes external content.

Threat in one sentence

Every external input the agent reads — fetched pages, files, channel messages, tool descriptions, sub-agent outputs — enters the same context window as the user's prompt and may carry instructions intended to manipulate the model's subsequent decisions.

L1 — Model resistance

- [ ] Use a frontier model with current-generation injection-resistance evals for any agent that acts on external content.
- [ ] Treat older or smaller models as a cost reduction and a safety reduction; document the trade-off.
- [ ] Where latency permits, use a two-model pipeline: a small fast classifier screens inputs before the action model sees them.
- [ ] Pin the model version. A silent provider-side model swap can change injection-resistance behaviour. (Chapter 8)

L2 — Strict-ignore guard system prompt

- [ ] Apply the strict-ignore guard verbatim to every agent's system prompt.
- [ ] The guard rejects: persona overrides, instruction modification, embedded directives, system-prompt disclosure, embedded code execution, and unsolicited URL fetches.
- [ ] Test the guard with adversarial inputs at deployment and after each model upgrade.
- [ ] Tailor the per-agent guard to the agent's role: "You are a research agent. You have no shell tool. Do not produce output that implies the existence of one."

L3 — Provenance and sanitisation

- [ ] Tag all fetched content as untrusted in the prompt — wrap in `<external_input source="...">...</external_input>` blocks.
- [ ] Use the SDK's structured roles correctly; never put fetched content in the user-prompt role.
- [ ] Pattern-strip known injection markers in fetched content: `SYSTEM:`, `ASSISTANT:`, `[INST]`, `</s>`, "ignore previous instructions", "new instructions".
- [ ] Truncate fetched content to a reasonable length; reject content with embedded URLs the agent would need to follow.
- [ ] For high-risk agents, run fetched content through a smaller screen model before the action model sees it.

L4 — Tool-side containment

- [ ] Every state-changing tool call passes through a Policy Enforcement Point (PEP) at call time.
- [ ] Tool scopes are narrow at the tool boundary, not at the prompt: allow-listed channels, allow-listed paths, allow-listed domains.
- [ ] Redact known secret patterns in tool arguments (PreToolUse hook) and tool outputs (PostToolUse hook) — see the Secret Management checklist.
- [ ] Irreversible actions (sending email, public posts, payments, deletions, deployments) require human-in-the-loop confirmation. Every time.
- [ ] If confirmation volume is impractical, the agent has too many sensitive capabilities; split it into more narrowly scoped agents.

Anti-patterns to remove

- [ ] System-prompt wording is the primary defence against injection. (It is a soft layer only.)
- [ ] Front-end input filtering only. (Indirect injection bypasses the front end.)
- [ ] Model-level injection benchmarks treated as a sufficient control.
- [ ] Sandboxed agents assumed unaffected by injection. (Sandbox limits consequences, not occurrence.)
- [ ] Conversation-log review treated as a substitute for prevention.
- [ ] Confirmation prompts assumed to be read.

Calibration by risk profile

Agent profile	Recommended layers
Research agent posting to a single channel	L1 + L2 + narrow tool scope
Coding agent with shell access	L1 + L2 + L3 + enforced PEP
Agent performing irreversible production actions	L1 + L2 + L3 + L4 + HITL on every such action

Adversarial review prompt: ”What is the most damaging tool call that could be induced from a fetched webpage?”