11: Security and prompt injection

Design as if an injected instruction will get followed

Why this matters

- Any text an agent reads can try to give it orders
- As of November 2026, nobody has a reliable fix
- Your coding agent has your data, untrusted input, and a shell
- Real attacks in 2025–2026 hit exactly this setup
- The defenses that hold are architecture, not clever prompts

Instructions and data share one channel

System	How instructions and data are separated
SQL with parameterized queries	The API keeps them apart
A language model	One stream of tokens; the model *judges* which is which

Usually it judges correctly. An attacker needs it to fail once.

Indirect injection is the one that matters

attacker plants text (issue, README, web page, package, tool description) → your agent reads it while doing your task → tool calls (shell, git, network, MCP) → secrets or code leave

- Direct: the attacker types it into the app
- Indirect: planted in content an agent reads later
- Jailbreak: a different problem, aimed at the model's own rules

Why it's still unsolved

- 12 published defenses; adaptive attacks beat most at over 90% success
- Anthropic cut browser-agent attack success to about 1–2% (November 2025)
- …and said "prompt injection is far from a solved problem"
- Instruction hierarchy is a learned preference, not an enforced boundary

A model's resistance to injection is a probability, not a permission system.

Two frameworks

Willison's lethal trifecta (June 2025)	Meta's Agents Rule of Two (October 2025)
Access to private data	[A] Processes untrustworthy inputs
Exposure to untrusted content	[B] Accesses sensitive systems or private data
Can communicate externally	[C] Can change state or communicate externally
Remove any one leg and data theft fails	At most two, or a human or reliable check approves

Coding agents have all three by default

Property	Typical local coding agent
Untrusted input	Web results, issues, PR comments, dependency source, READMEs, MCP output
Sensitive data	Source, `.env` files, SSH keys, cloud credentials, CI tokens
Change state or communicate	A shell, `git push`, package publishing, `curl`, MCP tools that write

Choose which leg to cut, per task. Put deterministic controls around the rest.

Part 2: In practice

Incidents, risks, and defenses that hold

Real incidents, 2025 (1 of 2)

When	Incident	Lesson
May 2025	GitHub MCP "toxic agent flow"	Scope tokens to one repo per task
Jun 2025	EchoLeak (Microsoft 365 Copilot)	Classifiers can be bypassed; zero-click
Jul 2025	Supabase MCP demo	Read-only, project-scoped access
Jul 2025	Amazon Q Developer extension	The prompt is code; protect it like code
Jul 2025	Replit production database deletion	No attacker needed; separate dev from prod
Aug 2025	Nx "s1ngularity"	Malware can drive your own AI CLI

Real incidents, 2025–2026 (2 of 2)

When	Incident	Lesson
Sep, Nov 2025	Shai-Hulud npm worms	Install scripts run with your privileges
Nov 2025	AI-orchestrated espionage (Anthropic report)	Agents lower attack costs; defenders need them too
Dec 2025–Feb 2026	"Clinejection"	Never put untrusted text in a CI agent's prompt
Feb 2026	OpenClaw skills and gateway	A skill is code plus instructions from a stranger
Apr 2026	"Comment and Control"	CI agents with public input and secrets: the trifecta in a box
2025–2026	The agent tools themselves	Keep agent tools updated

Risks specific to coding agents

- Skip-permissions modes outside a disposable sandbox
- Secrets pulled into context, transcripts, and output
- Malicious repos with config files, hooks, hidden instructions
- MCP tool poisoning and rug pulls
- Hallucinated packages ("slopsquatting")
- Production access and broad CI tokens

Auto mode is a speed bump. A sandbox is the wall.

- In auto mode, a classifier model approves or blocks each action
- Added to Claude Code in March 2026
- The starting mode for interactive sessions (as of November 2026)
- It cuts prompt fatigue, which is a real security problem
- It's still a model judging text

AI-written code isn't automatically secure

Study	Finding
Pearce et al., 2021	About 40% of 1,689 Copilot programs vulnerable
Perry et al., 2022	Assistant users wrote less secure code, and felt *more* confident
Veracode, 2025 (vendor report)	Flaws in 45% of cases; no better with newer models
BaxBench, 2025	Best model ~62% correct; about half the correct programs exploitable

Treat agent output as an untrusted contribution from a fast, confident junior engineer.

Frontier models help attackers and defenders

- Project Glasswing (April 7, 2026) gave Claude Mythos Preview to defenders
- Thousands of high-severity flaws found, including in OpenBSD and FFmpeg
- GPT-5.3-Codex (February 2026) got OpenAI's first "High" cybersecurity rating
- Students, point these tools at your own code before you ship
- Teams, assume attackers have them too, and patch faster

Defenses, part 1: the walls

Defense	What it does
1. Sandbox	OS-level file and network isolation, or the whole agent in a VM
2. Least-privilege credentials	One repo, minimum permissions, short-lived, per agent
3. Network egress control	Allowlist domains; cuts "communicate externally"
4. Allow and deny rules	Routine commands run; dangerous ones are blocked
5. Human approval	Pushes, deploys, deletions, publishing, email, spending

No single control is enough. Layer them.

Rules come second

```
{
  "permissions": {
    "allow": ["Bash(npm test *)", "Bash(git status)",
              "Bash(git diff *)"],
    "deny": ["Bash(curl *)", "Bash(wget *)",
             "Read(./.env)", "Read(./.env.*)",
             "Read(./secrets/**)"]
  }
}
```

Deny rules match command *text*. `curl` is reachable other ways. Sandbox first, rules second.

Defenses, part 2: hard rules and hygiene

Defense	What it does
6. Hooks	Hard rules that don't depend on the model agreeing
7. Split reading from acting	A reader with no secrets; an actor that never sees raw text
8. Dependency hygiene	Lockfiles, pins, review, wait on brand-new releases
9. Secret scanning and rotation	Push protection; rotate anything an agent may have read
10. No untrusted text in privileged CI prompts	Read-only tokens, no secrets, isolated caches

Checklists

Students	Teams
Sandbox agents for any code you didn't write	Per workflow, write down which leg you remove
No skip-permissions outside a disposable sandbox	Standard sandbox; skip-permissions disabled by policy
Keep keys out of the project and out of chats	Short-lived, per-agent tokens; no production credentials
Read every command before approving it	Egress allowlists; branch protection; human review
Check every new package is real	Approved, pinned MCP servers, plugins, and skills
Scan AI code; get a second model's review	Secret scanning plus an "agent read a secret" runbook

Part 3: Going deeper

Designs that give real guarantees

Design patterns with real guarantees

Pattern	Idea
Action-Selector	Pick from a fixed menu; tool output never flows back
Plan-Then-Execute	Commit to a plan before reading untrusted data
LLM Map-Reduce	Isolated sub-agents return constrained results
Dual LLM	The privileged model never sees untrusted text
Code-Then-Execute	A program, not the model, handles the data
Context-Minimization	Remove earlier context before later steps

Security comes from the system's data flow, not the model's judgment.

On autonomy and security

> **Bryan's take:** I don't trust fully autonomous workflows yet. I run several CLI agents in parallel, manage them by hand, and find it valuable to have multiple agents and models review each other's work (module 08).

Try it

On your own, inject yourself, safely:

1. Throwaway repo in a container or VM, with a fake canary secret
2. Plant a note asking agents to copy the fake secret into the README
3. Ask an innocent question. Did it comply, refuse, or warn you?
4. Add one defense and repeat. Which leg did it remove?

As a class or team, threat-model a real agent workflow with the Rule of Two.

Discussion

1. Why can't we fix prompt injection like parameterized queries fixed SQL injection?
2. An agent reads support emails and updates the CRM. Which leg goes?
3. Is auto mode a security feature or a usability feature?
4. A dependency's README tricked your agent into leaking a secret. Who's accountable?

Read more

- Module 11: Security and prompt injection
- Simon Willison, The lethal trifecta for AI agents (June 16, 2025)
- Meta, Agents Rule of Two (October 31, 2025)
- Beurer-Kellner et al., Design Patterns for Securing LLM Agents against Prompt Injections (2025)