

Failure Analysis #1

# Why Prompt-Level Safety Fails at the Irreversible Action Boundary

Prompt injection and related adversarial control attacks defeat conventional safety layers because safety often lives in language space without execution authority. Execution-time governance changes the attack surface at the irreversible action boundary, where text becomes state change.

Audience	Senior engineers; safety architects; AI/autonomy decision-makers
Scope	Architectural failure analysis (no exploit recipes; no UI/code)
Core claim	Prompt injection succeeds when safety lives in a channel without execution authority.
Non-claim	Execution-time governance improves irreversible-harm prevention; it does not “secure” the model.
Date	2026-01-21

**Authoritative inputs** Internal execution-time evidence artifact (2026-01-21) plus public literature on prompt injection, tool-using agents, jailbreaks, and enforcement mechanisms.

# Table of Contents

Table of Contents	2
1. Problem Framing	3
2. Why Conventional Safety Layers Fail	4
2.1 Layer-by-layer: why this is not execution authority	4
2.2 The irreversible-action failure mode	4
3. Shift in Control Surface (the key insight)	5
4. Why Execution-Time Governance Changes the Game	6
4.1 Internal evidence pattern (internal artifact)	6
4.2 Why the difficulty multiplies (not just adds)	6
4.3 Why this is not 'just guardrails'	7
5. What This Does Not Claim	7
6. Conclusion	8
References	9

# 1. Problem Framing

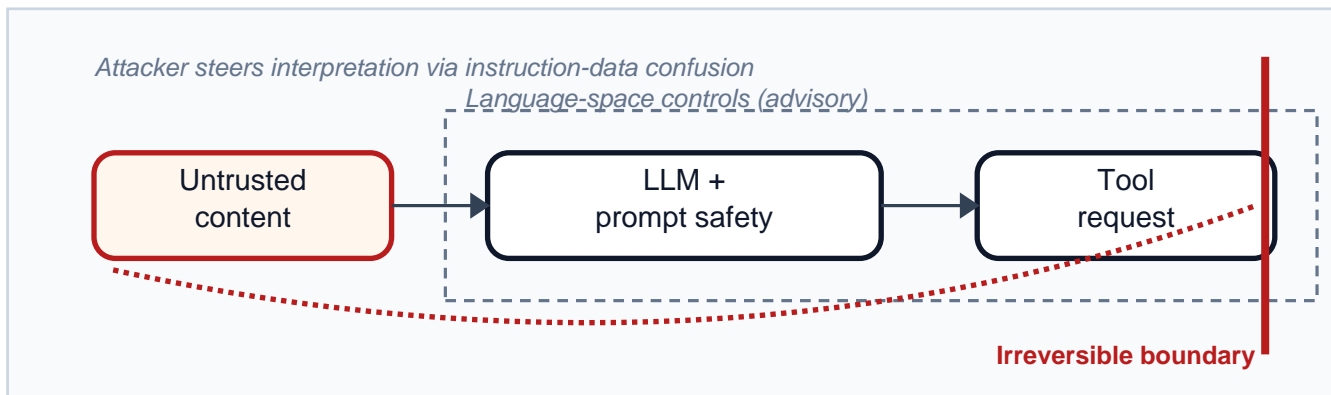
Prompt injection and tool-misuse attacks do not “hack the model weights”; they exploit a more mundane failure: systems treat *untrusted language* as if it were a privileged control channel. In LLM-integrated applications, instructions and data are often co-resident in the same context, so an attacker’s content can be interpreted as higher-priority guidance. This is particularly acute when the model can call tools or trigger workflows, because natural language becomes the de facto control surface for external actions.

- **Attack target:** the system’s *authority mapping* (which inputs can cause which actions), not “content safety”.
- **Assumed authority location:** prompts, policies, and model-side refusal behavior, rather than an external enforcement layer.
- **Why it works:** attackers use the same medium as legitimate control (text), creating instruction-data ambiguity the model has no formal way to resolve.

Security researchers describe this as a modern instance of the *confused deputy* problem: a powerful component (the model + tool permissions) can be induced to misuse its authority on behalf of an untrusted party [6]. Prompt injection is also argued to be structurally different from SQL injection because LLMs lack a native, enforceable separation between “instruction” and “data” [7].

## Figure 1. Attacker vs intent-based (prompt-governed) system

*Authority is inferred from text; safety remains advisory before the irreversible boundary.*



**Key boundary:** the irreversible action boundary is where text becomes state change (sending money, executing code, modifying records, placing orders). If safety does not control that boundary, it is at best advisory.

## 2. Why Conventional Safety Layers Fail

Conventional safety layers fail at the irreversible action boundary because they can influence behavior but cannot authoritatively prevent execution. Most operate upstream of commit: they shape what the model says, then hope downstream systems comply. Once an attacker can steer interpretation of context, that hope becomes the failure mode.

### 2.1 Layer-by-layer: why this is not execution authority

Layer	Typical mechanism	Why it lacks execution authority (structural)
Prompt-level controls	System prompt; instruction hierarchy; templates	Lives in the same channel as attacker content; cannot guarantee instruction-data separation; persuasion rather than mediation [7,9].
Policy classifiers	Refusal models; content filters; jailbreak detectors	Decisions are made from text features (or model outputs), not authoritative state; adversarially brittle under distribution shift and adaptive prompting [3].
Tool permissioning	Allowlist tools; scoped API keys; per-tool constraints	If the model can still request a permitted tool, permissioning only bounds the menu. Confused-deputy behavior remains possible [6].
Monitoring / alerts	Logs; anomaly detection; human review of outcomes	Observes outcomes but does not mediate them; cannot satisfy complete mediation or fail-safe defaults at the action boundary [5,8].
Human approval	Manual confirmation; “are you sure?” prompts	Humans sit outside the commit path and can be socially engineered or bypassed under latency pressure; approval is advisory unless embedded in the commit mechanism [5].

**Note on “monitoring-only” safety.** In classical security engineering, an enforcement mechanism should be invoked on every security-relevant access (complete mediation) and should be tamper-resistant. Purely observational mechanisms cannot provide these properties. This is why monitoring cannot be the primary defense at an irreversible action boundary [5,8].

### 2.2 The irreversible-action failure mode

At the irreversible boundary, a system must answer: “Is this action permitted *given the current authoritative state*?” Prompt-level and classifier-based layers instead answer: “Does this text look allowed?” These are not equivalent questions. The gap is where prompt injection and similar attacks accumulate leverage (especially in tool-using agents) [1,2].

### 3. Shift in Control Surface (the key insight)

The central architectural insight is that **intent is not authority**. Conventional safety layers often conflate the two by treating a model's expressed intent (in text) as a proxy for what the system will do. The moment a system can act through tools, the real question is which component has the power to commit state.

- **Intent  $\neq$  authority.** A model may intend to comply or refuse, yet still produce an action proposal that triggers execution via a downstream adapter.
- **Belief  $\neq$  validity.** A model can be convinced a premise is true; validity requires external state checks (permissions, invariants, freshness).
- **Language  $\neq$  state.** Text descriptions of the world are not the world; enforcement must reason over authoritative state, not narrative.
- **Decision  $\neq$  execution.** A decision is a proposal until an execution layer commits it. The component that commits is the authority boundary.

Recent research frames instruction-data separation as a fundamental architectural challenge rather than a prompt-engineering nuance. Work on prompt injection and on instruction/data separation argues that LLMs lack a principled internal distinction between passive data and controlling instructions, so “prompt rules” alone cannot deliver robust guarantees [7,9].

Treat the model as an *untrusted proposer* and enforce safety at the point of action. This mirrors reference-monitor and capability-based thinking: authority should be explicit, minimal, and mediated on every security-relevant access [5,6].

## 4. Why Execution-Time Governance Changes the Game

Execution-time governance changes the attack surface by relocating authority from “what the model says” to “what the system is allowed to do right now”. It does not attempt to make adversarial intent impossible. It makes adversarial intent insufficient to cross the irreversible action boundary.

### 4.1 Internal evidence pattern (internal artifact)

The internal execution-time evidence artifact exhibits a consistent execution-governance shape: upstream components generate candidate actions, but a downstream authority layer mediates and can fail closed. The evidence includes: (i) explicit “final authority” rules, (ii) stale/missing authoritative-state checks that block execution, and (iii) mode/role gates (e.g., shadow/read-only) that prevent writes even if a decision exists.

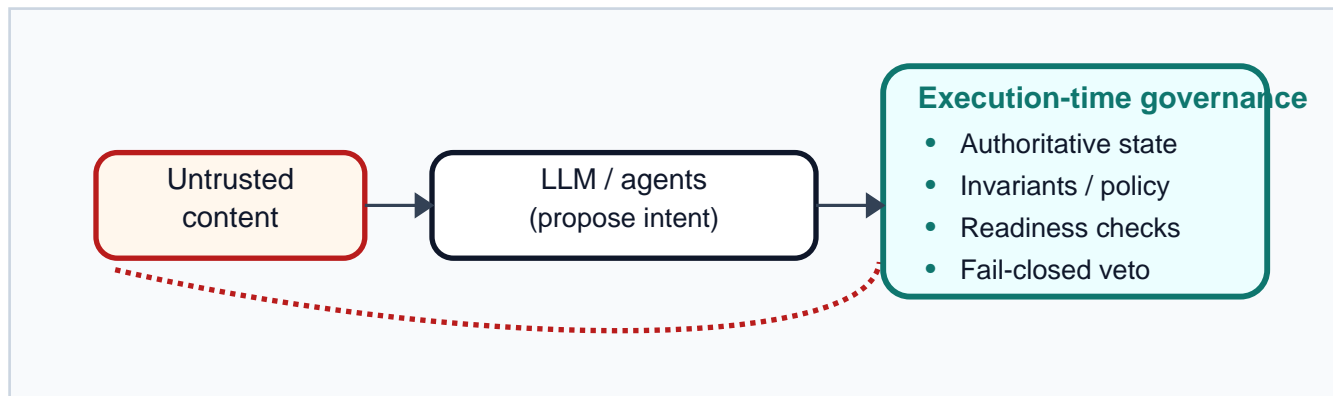
#### Illustrative governance predicates (from internal evidence):

- “Risk has final authority; if risk rejects, no trade.”
- Fail-closed on missing/stale/ambiguous inputs.
- Execution disabled in shadow mode or restricted modes; write attempts are blocked.
- Execution blocked if authoritative account state is missing/stale/incomplete.

Not monitoring. **Execution vetoes.**

### Figure 2. Attacker vs execution-governed system (runtime authority)

*The attacker must satisfy governance predicates, not merely persuade the model.*



### 4.2 Why the difficulty multiplies (not just adds)

Difficulty multiplies because the attacker must defeat multiple independent predicates - many grounded in state the attacker cannot directly control. In an intent-based system, the attacker mostly needs to win a single contest: persuade the model (or the prompt/classifier wrapper) to emit an action proposal. In an execution-governed system, persuasion is only the entry ticket.

- (a) **Authoritative state** (permissions, freshness, identity, resource constraints).
- (b) **Invariants** (safety constraints expressed as state predicates, not text rules).
- (c) **Execution readiness** (preconditions, mode capabilities, dependency health).
- (d) **Fail-closed veto** (absence/ambiguity blocks action by default).

Even if the model is fully “jailbroken” at the language layer [3], these gates can still prevent irreversible harm, because they are not implemented as language behavior.

This maps to established security design principles: complete mediation and fail-safe defaults demand that the enforcement point sees every action attempt and denies by default when uncertain. Execution-time governance instantiates those principles for tool-using AI systems [5,8].

### 4.3 Why this is not ‘just guardrails’

Guardrails are often implemented as prompt wrappers or post-hoc filters. Execution-time governance is different in kind because it is authoritative: it sits on the only path that can commit irreversible state, and it can veto regardless of what the model claims or believes. This is the architectural distinction a patent framing should capture: authority is relocated to an execution monitor invoked on every security-relevant operation.

## 5. What This Does Not Claim

- **Does not eliminate adversarial inputs.** Attackers can still attempt to manipulate prompts, retrieved data, or intermediate agent messages [1,2].
- **Does not make models “secure”.** Models can still produce unsafe text, leak information, or degrade reliability; governance targets irreversible action prevention, not universal correctness.
- **Does not solve all AI safety.** It does not address value alignment, deception, or long-horizon autonomy in general; it addresses a narrower control problem: preventing unauthorized state transitions.
- **Does not remove the need for monitoring.** Observability remains necessary for incident response and assurance, but it is insufficient as a primary control at the action boundary [5].

Execution-time governance constrains harm even when upstream behavior is compromised. **Strong property.** Not invulnerability.

## 6. Conclusion

Prompt injection and adjacent adversarial control attacks succeed because conventional safety lives where authority does not. Prompt rules, classifiers, and monitoring can influence what the model says, but they cannot reliably control what the system does at the irreversible action boundary. Execution-time governance changes this by making the decisive question stateful and enforceable: whether an action is permitted given current authoritative state, with fail-safe defaults and complete mediation on every attempt.

**Series thesis.** Safety mechanisms that do not possess execution authority are structurally lossy. The only durable way to prevent irreversible harm is to move authority to the execution point, enforce invariants over authoritative state, and fail closed when uncertain.

This framing is compatible with public findings that (i) instruction-data separation is architecturally hard [7,9], (ii) prompt injection scales in tool-using agents [1,2], and (iii) purely prompt-level defenses can be bypassed by automatically discovered adversarial prompts [3]. The internal execution-time evidence provides an existence proof that fail-closed, stateful vetoes are implementable in real systems and can be made authoritative at execution time.



# References

[1] K. Greshake et al. ‘Not what you’ve signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection.’ arXiv:2302.12173, 2023; also ACM CCS 2023.
[2] E. Debenedetti et al. ‘AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents.’ arXiv:2406.13352, 2024 (NeurIPS Datasets & Benchmarks submission).
[3] A. Zou et al. ‘Universal and Transferable Adversarial Attacks on Aligned Language Models.’ arXiv:2307.15043, 2023.
[4] OWASP. ‘Top 10 for Large Language Model Applications.’ (LLM01 Prompt Injection, etc.), accessed 2026-01-21.
[5] J. H. Saltzer and M. D. Schroeder. ‘The Protection of Information in Computer Systems.’ Proceedings of the IEEE, 63(9), 1975.
[6] N. Hardy. ‘The Confused Deputy (or why capabilities might have been invented).’ ACM SIGOPS Operating Systems Review, 22(4), 1988.
[7] UK National Cyber Security Centre (NCSC). ‘Prompt injection is not SQL injection (it may be worse).’ Blog post, 2025-12-08.
[8] F. B. Schneider. ‘Reference Monitors.’ (enforcement mechanisms chapter / notes on complete mediation, tamperproofing), 2023 revision.
[9] E. Zverev et al. ‘ASIDE: Architectural Separation of Instructions and Data in Language Models.’ arXiv:2503.10566 / ICLR 2025 version, 2025.
[10] Internal execution-time evidence artifact (2026), 2026-01-21.