OWASP LLM TOP 10 · LLM01

Prompt Injection

Why it's structurally unsolved — and how to defend anyway

A senior-level deep dive · SWE/AI Learning Tracker

THE ROOT CAUSE

Where This Problem Starts: SQL Injection

Code and data merged into ONE parsed string

```
SELECT * FROM users
WHERE name = '" + input + "'
```

```
input = "'; DROP TABLE users; --"
```

The parser can't tell the difference between
the query's structure and the attacker's payload.

One channel

Structure & data share a single string

No boundary

Nothing marks where data ends

Reparsed as code

Attacker payload executes

THE FIX

Parameterized Queries: A Hard Guarantee

01

Parse structure first

The query shape is compiled before data arrives

02

Bind data second

Values are inserted, never re-parsed

03

Guarantee, not habit

Enforced by the parser — outside any model or judgment call

THE CORE INSIGHT

LLMs Have No Equivalent Guarantee

Role-separator tokens exist...

...but "user tokens = lower authority" is a

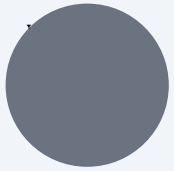
learned statistical pattern in the weights

not a mechanically enforced parser rule.

It's a habit the model tends to follow — not a boundary anything outside the forward pass enforces.

FRAMING IT CORRECTLY

“Hard to Patch” vs. “Structurally Unsolved”



Hard to patch

(wrong framing)

Implies: enough filters/rules eventually close the gap

Structurally unsolved

(correct framing)

No mechanism outside the model's forward pass gates the final action

Defensive posture: assume injection sometimes succeeds — design guardrails for that, not perfect prevention.

OWASP'S TAXONOMY

Direct vs. Indirect Injection — Split by Channel

Direct

Attacker talks straight
to the model in the prompt

Indirect

Instruction rides in on
third-party content the model reads

Defined by CHANNEL — not attacker identity, not victim awareness

A user hiding a jailbreak in their own doc, aimed at the model itself, is still “indirect” by this definition.

WHERE THE HARM REALLY IS

The Confused Deputy Problem

NOT this:

"User data can contain commands" — that's normal product use (e.g. "read my doc, follow its rules")

THIS:

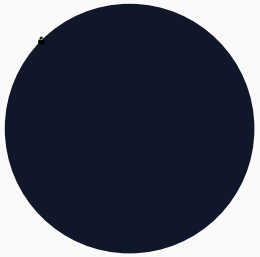
Third-party content carries instruction-authority the user never delegated to it.

User (the principal) | Model (the deputy) | 3rd-party content (the impostor)

The deputy (model) confuses the impostor's voice for the principal's — and acts on its behalf.

AN EDGE CASE THAT BREAKS THE SIMPLE FRAME

When the User Delegates On Purpose



“Read my Google Doc and follow its rules.”

The user consciously hands instruction-authority to their own data.
That's consent — not a confused deputy.

Consent is a SEPARATE, orthogonal axis

It doesn't change the direct/indirect channel boundary — it just explains why some “indirect” instruction-following is fully legitimate rather than an attack.

ANATOMY OF A DEFENSE

Three Failure Points

Boundary detection

Is this content third-party?

Mostly solved

Conflict detection

Does it fight the system prompt?

Weak for subtle cases

Enforcement

Can the action be stopped?

Structurally unsolved

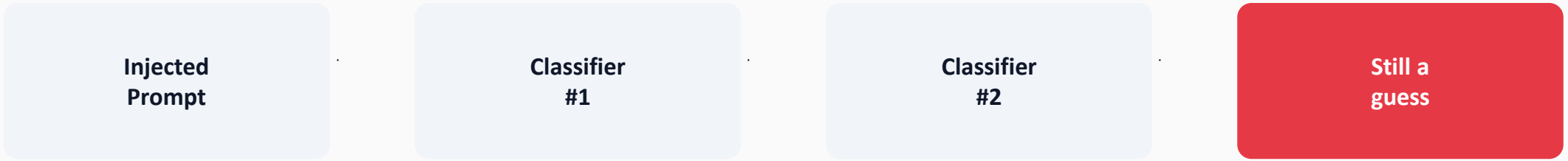
THE SHARPEST INSIGHT

Detection ≠ Enforcement

Detection is made of the same stuff as what it detects.

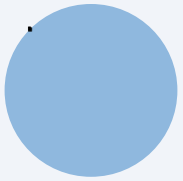
An LLM-based classifier is just another learned habit.

Stacking classifiers shifts the failure mode down a level — it doesn't eliminate it:



WHERE HARM ACTUALLY LANDS

Bad Output vs. Bad Action



Bad Output

Harmful or leaked text

Caught reliably by output scanners

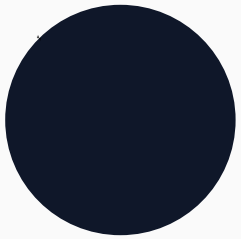
Bad Action

A tool call driven by injected intent

Looks authorized — scanners largely miss it

THE PARTIAL FIX FOR BAD ACTION

Provenance / Taint Tracking



Label trust at ingestion, trace it through to the action.

Only an APPROXIMATION — not a true causal trace

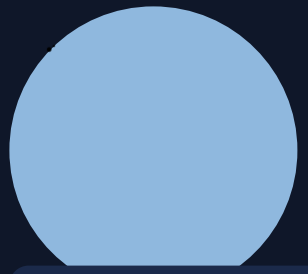
LLMs have no call stack, no mechanical record of which context tokens caused a decision — the forward pass is diffuse.

Fallback in practice: label trust levels upfront (spotlighting) rather than tracing causation after the fact.

Real systems: dual-LLM patterns, CaMeL-style capability tracking

MITIGATION 1 OF 6

Privilege Limits (Least Privilege)



Risk classification lives OUTSIDE the model — in code, at the tool-execution boundary.

Strength

A closed, enumerable space — tool call is either in the allowed set, or it isn't. Injection can never talk its way past it.

Trade-off

Only as strong as what's explicitly gated. Blocks legitimate broad-access tasks too.

MITIGATION 2 OF 6

Human-in-the-Loop

A human approves the action before it fires — complementary to privilege limits.

Failure mode: approval fatigue

Repeated low-stakes prompts degrade into reflexive rubber-stamping — the human stops actually checking.

MITIGATION 3 OF 6

Input / Output Filtering

Open, infinite space of natural-language meaning — needs semantic judgment, not lookup.

Input filtering

Attacker authors it directly.
A head-to-head arms race against known filters.

Arms race

Output filtering

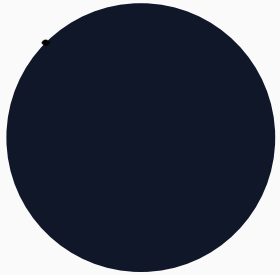
One extra hop: attacker → injected instruction → model → output.

Speed bump, not a wall

Still: effective vs. bad output (detectable pattern) — blind to bad action (a clean, authorized-looking call).

MITIGATION 4 OF 6

Spotlighting / Delimiting



Finer-grained boundary marking — sub-turn, not turn-level.
Extends the role-separator idea.

Adds nothing to enforcement

Nothing outside the forward pass —
same habit-not-guarantee ceiling.

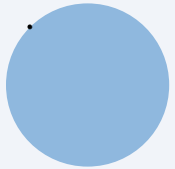
Improves conflict detection

Sharpens the model's pattern-matching
to locate which spans deserve scrutiny.

MITIGATION 5 OF 6

Schema-Constrained Output

Privilege-limiting, applied to the output side.



A real, closed-space, structural check — a field either exists in the schema or it doesn't.

Limited to bad action only — field content is still open natural language, still needs semantic judgment for bad output.

MITIGATION 6 OF 6

Adversarial / Red-Team Testing

A PROCESS, not a runtime control.

Continuously probes for new injection patterns before attackers find them — feeds every other mitigation, but never fires at inference time.

PUTTING IT ALL TOGETHER

The Full Mitigation Map

Mitigation	Target	Guarantee
Privilege limits	Enforcement	Hard
Human-in-the-loop	Enforcement	Hard*
Input filtering	Conflict detection	Soft
Output filtering	Conflict detection	Soft
Spotlighting	Conflict detection	Soft
Schema-constrained output	Enforcement (action only)	Hard
Red-team testing	All (process)	N/A

**Hard on the gate itself; still degrades in practice via approval fatigue.*

REVIEW QUEUE FINDING

Taxonomy Gap: Channel, Not Identity

Direct vs. indirect is defined by CHANNEL — not attacker identity, not victim awareness.

A user hiding a jailbreak in their own document/tool-output, targeting the model itself, is still INDIRECT injection.

Even though attacker = user. A standard red-team technique — it evades defenses tuned for direct-injection phrasing.

The "consent to delegate" exception (slide 8) is a separate, orthogonal axis about consent — not part of this boundary.

LLM01 — RECAP

Prompt Injection, In One Slide

Habit, not guarantee

Role authority lives in weights, not a parser

Detection ≠ Enforcement

Classifiers are made of the same stuff they detect

Bad output vs. bad action

Scanners catch text, miss clean-looking tool calls

Enforce outside the model

Privilege limits + schema constraints = real guarantees

Next up

Agent Frameworks (LangGraph / CrewAI)

Then resume LLM02–LLM10 across the rest of the OWASP LLM Top 10.