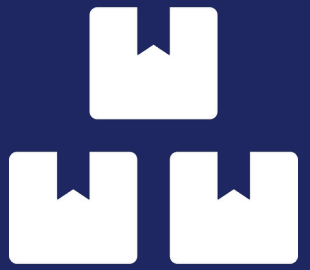


LLM04:2026

Supply Chain

OWASP Top 10 for LLM Applications — 2026

Why It Matters Now

40,000+

AI pipelines exposed

40 min

breach window

36%

of cloud environments hit

March 2026: attackers compromised LiteLLM (widely-used LLM proxy gateway) via a single poisoned PyPI package — harvesting cloud credentials, API keys, and Kubernetes secrets without a single prompt being touched.

Now formalized as its own attack class: **OWASP Agentic Top 10 ASI04 — Agentic Supply Chain Vulnerabilities.** Even frontier-lab tooling isn't exempt: Claude Code's own hooks/config shipped CVE-2025-59536 (Feb 2026).

Two Distinct Risk Shapes

1. The Model Itself

Deliberately tampered with

Behaves correctly on every normal input — passes eval, passes benchmarks — and only misbehaves on a secret trigger. Not a bug. Intentional, hidden behavior baked into the weights.

2. Surrounding Infra

Third-party tooling compromised

The model is fine. A dependency isn't — a proxy, an MCP server, a plugin. Same shape as classic software supply-chain attacks, applied to AI-specific tooling.

Pre-Deployment: Check Provenance

Same instinct as vetting an npm package's maintainer/download history — applied to a model repo instead of a code repo.

- Verified / known publishing org (not an anonymous uploader)
- Signed or verified-badge checkpoint, where the hub supports it
- Model card discloses training data and intended use
- Download count and community scrutiny — is anyone else relying on this?

Provenance Reduces Risk — It Doesn't Eliminate It

Trusted account gets hijacked

Same shape as the LiteLLM maintainer compromise — a reputable org's credentials or publishing pipeline is breached, not the org itself acting maliciously.

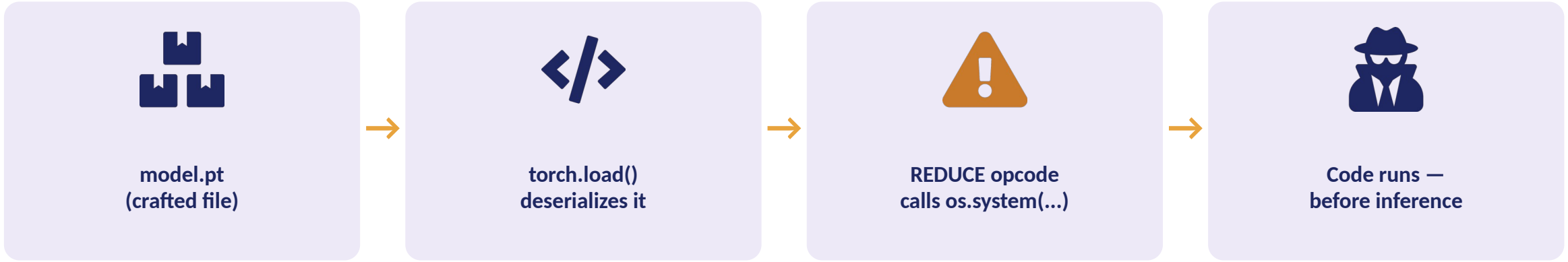
Poisoned upstream training data

Even a well-intentioned org can unknowingly train on tampered data (scraped web content, compromised fine-tuning sets), baking in a backdoor nobody put there on purpose.

The necessary second layer: **independent evaluation.** Run your own red-team/adversarial suite against the model's actual behavior — the only check that looks at what the model does, not who says they made it.

The More Common Exploit: Pickle Deserialization

A .pt / .pkl checkpoint isn't just tensors — it's a serialized Python object graph.



Fix: safetensors — stores only raw tensor data, no executable opcodes. Loading it structurally cannot run arbitrary code. Hugging Face now flags non-safetensors uploads for exactly this reason.

Adapter / LoRA Merging Risk

$$W_{new} = W + A \times B$$

A small, low-rank correction (megabytes, not gigabytes) merged into — or applied alongside — the frozen base model.

- Widely shared, thousands per hub — far less scrutiny than a full base model
- Carries the same pickle-RCE and backdoor-trigger risk as any checkpoint
- Easier to hide a narrow, targeted misbehavior in — fewer weights to tamper with
- Merging collapses the trust boundary — once baked in, you can't tell which part of the final weights came from the unaudited adapter

Closing Take

“You can’t code-review a tensor.”

Code-level supply chain security (diff, audit, sign) breaks down against opaque model weights — defenses shift to provenance tracking, independent evaluation, and adapter/dependency scrutiny instead of source inspection.

- **Provenance checklist**
- **Independent red-team eval**
- **safetensors over pickle**
- **Adapter/LoRA scrutiny**
- **License review**