5 Ways AI Breaks Threat Modeling

A practical checklist for security teams deploying agentic AI

> Most engineering teams deploying AI in 2026 already have a security process. STRIDE reviews, threat registers, penetration tests before major launches. That process was not designed for AI. AI breaks all three of those assumptions: outputs are not deterministic, the attack surface includes training data, and agents take actions beyond producing text. Here are the five specific ways, with what you need to add before you ship.

1 Outputs Are Probabilistic, Not Deterministic

You cannot unit-test a distribution with a single pass. Your test suite has no predictive power over behavior it did not observe.

☐ Automated statistical evaluation Run thousands of attack-prompt variations and report attack success rates with confidence intervals, not binary pass/fail results.

WHY *A prompt reliably refused yesterday may be accepted today after a model update, a temperature change, or a shift in conversational context.*

☐ Re-testing on a model-update schedule Trigger re-tests whenever the model, prompt templates, or tool definitions change, not just at pre-deployment.

WHY *A financial services team found zero bypasses in testing. Days after launch, a user reported a bypass caused by a model update between testing and production. Every test passed. None predicted the production behavior.*

☐ Behavioral baselines with alerting Record output distributions and key behavioral metrics during staging. Alert when production behavior deviates significantly.

WHY *You cannot detect drift without a baseline. Silent behavior shifts are the most common source of surprise in AI deployments.*

☐ What changes STRIDE categories still apply to AI systems, but each maps to a different attack class. Probability replaces binary outcomes. Confidence intervals replace pass/fail. Continuous monitoring replaces pre-deployment snapshots.

2 The Attack Surface Includes Training Data

You can audit source code and dependencies. You cannot audit a learning process that has already concluded.

☐ Corpus provenance documented Record sources, licenses, collection dates, and responsible parties for all training and fine-tuning data.

WHY *Undocumented data makes compliance, legal, and security reviews impossible to complete honestly.*

☐ Poisoning resistance evaluated Run statistical anomaly detection on training corpora. Test with membership inference attacks. Verify differential privacy guarantees if applicable.

WHY *Data poisoning embeds persistent backdoors that survive fine-tuning. No CVE, no visible exploit, no observable breach at the time of attack.*

☐ Access controls on data stores Restrict write access to training data, fine-tuning datasets, and embedding stores. Treat them like source code repositories.

WHY *A healthcare company fine-tuned a model on de-identified patient records. Membership inference attacks allowed researchers to identify which records were in training. The model was operating exactly as designed. The design was the vulnerability.*

☐ Indexed content treated as untrusted Code repositories, wikis, and external corpora feeding retrieval systems are untrusted input. Never grant them instruction-level authority.

WHY *Trusted-by-default indexed content is the primary vector for indirect prompt injection attacks.*

☐ What changes Your asset inventory now includes training corpora, fine-tuning datasets, and embedding stores. Each needs provenance documentation, poisoning resistance evaluation, and access controls. These do not fit on a traditional data flow diagram because the attack surface is a learning process, not a boundary.

3 Agents Take Actions, Not Just Produce Output

A standalone LLM producing harmful text is a content moderation problem. An agent acting on that text is an operational security problem.

☐ Tool allowlist verified The set of tools available to the agent is explicit, minimal, and reviewed. No implicit or ambient tool access exists.

WHY *An agentic customer support system was manipulated through a customer message. The model dispatched a tool call with attacker-supplied parameters. No code vulnerability was exploited. The vulnerability was in the gap between what the attacker said and what the model thought it was being asked.*

☐ Every tool parameter schema-validated Tool inputs are validated against a strict schema before execution. Malformed inputs are rejected, not coerced.

WHY *An agent can be tricked into constructing malicious tool inputs. Schema validation is a hard gate that the model alone cannot provide.*

☐ Credentials scoped per tool Each tool uses a dedicated, least-privilege credential. No broad tokens or shared secrets.

WHY *A single compromised credential should expose only one tool scope, not the entire environment.*

☐ Tool call audit trail captured Every tool invocation is logged with full input parameters and outcome in a tamper-resistant store.

WHY *Without a complete audit trail, incident investigation and compliance review are impossible.*

☐ What changes Every tool your agent can call is attack surface. You need a tool allowlist (not a denylist), parameter validation on every input, scope-limited credentials per agent, and full logging of every action. These do not appear on traditional threat modeling checklists because traditional systems do not have tools.

4 Prompt Injection Has No Equivalent Fix

SQL injection was solved at the parser. Parameterized queries tell the database: this is code, this is data. Language models have no parser.

☐ Input sanitization and structural separation System instructions and user-supplied data are structurally separated. The model is never asked to treat data as instructions.

WHY *Models that cannot distinguish instructions from data are trivially injectable. Structure is the first defense.*

☐ Untrusted content clearly delimited Retrieved content from web pages, documents, or repositories is wrapped in explicit delimiters. The model is instructed never to follow instructions within them.

WHY *Indirect injection attacks embed instructions in retrieved content. Delimiting is the primary mitigation.*

☐ Output validated against policy before action Agent outputs are validated against a declared schema and policy before any action or tool call is executed.

WHY *A compromised model may produce plausible but malicious outputs. Validation is the last gate before action.*

☐ Privilege boundaries hold under full model compromise Authorization checks live outside the model. The system does not rely on the model refusing a request to enforce access control.

WHY *If the model is compromised, it will not refuse. The surrounding system must enforce all boundaries independently.*

☐ What changes Defense-in-depth replaces single-point mitigations. Input sanitization, output validation, constrained tool permissions, behavioral monitoring. Each layer reduces exposure. None eliminates it. Your security design needs to account for residual prompt injection risk across the system's operational lifetime.

5 The Supply Chain Extends Beyond Code

Traditional supply chain security has a defined scope: source code, dependencies, build artifacts, container images. AI adds datasets, model weights, MCP servers, and skill marketplaces.

☐ MCP server scanning Run static analysis on tool descriptions looking for suspicious patterns: references to unexpected system paths, credential access, outbound network calls, or instructions conflicting with the tool's stated purpose.

WHY *A malicious MCP server was published to a community registry. Tool descriptions contained embedded instructions. The model executed them via tool calls. No vulnerable code was deployed. The exploit was a text string in a configuration file.*

☐ Tool description hash verification Hash tool descriptions and compare against known-good baselines. Any change, even a single word, triggers a security review before the agent loads the updated server.

WHY *As of mid-2026, no dedicated scanning tool exists for MCP tool descriptions. Most teams rely on manual review, which is precisely why this attack class is so dangerous.*

☐ Sandboxed MCP execution Run MCP servers in isolated environments with network egress controls. Treat them like untrusted plugins.

WHY *An untrusted MCP server can exfiltrate data, execute arbitrary commands, or modify other tool configurations. Sandboxing bounds the blast radius.*

☐ Extended SBOM coverage Extend your software bill of materials to cover datasets, model weights, embedding stores, and MCP servers. Standard SCA tools do not see prompt-level payloads.

WHY *The components that matter most to your AI system's security are the ones you cannot find in your dependency tree.*

☐ What changes Your pre-deployment process needs MCP server scanning, tool description hash verification, sandboxed execution, tool call monitoring, and extended SBOM coverage. The supply chain for AI systems is not a boundary you can audit systematically. It is a living ecosystem of text, weights, and configurations.

6 What This Means for Your Process

None of this means STRIDE is wrong. It means STRIDE is not sufficient.

☐ Existing practice is the foundation You still need threat modeling, penetration testing, and vulnerability management. These five gaps require an additional analytical layer on top that your current tools were not built to see.

WHY *If your AI security review for a production agent deployment looks identical to your review for a traditional API service, you have a blind spot. Not because you made an error. The threat surface includes components and behaviors your process was never designed to evaluate.*

Get the next one. aminrj.com/subscribe