AI PRODUCT SECURITY IN THE AGE OF MYTHOS · 2026

Chapter 12 · Model, Code, and AI Supply Chain Security

Standalone reading module for LMS delivery and required reading.

FORMAT	USE	SCOPE	AUDIENCE
Standalone PDF	Required reading	Single chapter	Learners

CHAPTER 12

Model, Code, and AI Supply Chain Security

LLM03

AI SUPPLY CHAIN

OWASP places LLM supply-chain risk in the 2025 Top 10. NIST SSDF and SLSA frame the response as dependency, build, provenance, and tamper-resistance evidence.

OWASP GENAI SECURITY PROJECT · NIST SSDF · SLSA

The model is not outside the supply chain. The model is one of the supply chain's most privileged participants.

AI product security has to govern three supply chains simultaneously: software (code, dependencies, containers, CI/CD), model and data (weights, endpoints, datasets, embeddings), and agent infrastructure (prompts, tools, plugins, workflows).

The Experiment That Became Production

A model artifact can become a production dependency faster than the security program notices.

A team in the office-productivity group downloads a model from Hugging Face to test for an internal chatbot. The model appears in a GitHub repository. It works well enough. The infrastructure team sees it is useful and wraps it in an internal service endpoint. The customer-support group begins using the endpoint as a proof-of-concept. Customer feedback is positive. Management approves expanding the POC to a small customer cohort. Six months later, the model is serving real customers. No one has ever recorded: the model's source repository, the exact version or hash, the training data or license, who authorized the deployment, what evaluation results justified the risk, how to roll back if the model fails, or who owns it if something goes wrong.

The model is now running in production. It influences customer interactions. It may influence billing decisions, compliance configurations, or data handling. But the organization has no supply-chain record for it.

The problem is not that open models are inherently unsafe. The problem is that an artifact moved from experiment to product without the supply-chain controls required by its risk.

> ***Models are supply chain artifacts, not experimental sidecars:*** *A production model needs provenance, version control, ownership, and a rollback path—the same controls required for any production dependency.*

If the organization cannot answer these questions, the model is inside the product but outside control.

This matters because model behavior can change. A new version may have different refusal patterns. A fine-tuned version may have different output distribution. A replaced version may fail in ways the old version did not. Without provenance and version control, the organization cannot reason about what is running or what went wrong.

The same principle applies to embeddings, adapters, LoRA weights, and any other weight artifact that shapes product behavior.

Prompt Packages as Product Behavior

> ***A prompt is not just text—it is behavior:*** *Prompts define refusal patterns, escalation logic, tool selection, approval gates, and logging. A prompt change can alter product behavior as meaningfully as a code change.*

Prompts may define: what the model refuses, what it escalates, how it selects tools, what format it outputs, when it asks for approval, what it logs, and how it handles error conditions. In some products, a prompt change alters behavior as meaningfully as a code change.

A support platform changes its prompt from "Be conversational and helpful" to "Prioritize speed. Resolve in one message if possible." The model now escalates less frequently to humans. Complaint volume rises. No code changed. No model changed. A prompt line changed behavior.

A developer agent's prompt shifts from "Always ask for approval before pushing code" to "Ask for approval only if changes affect core authentication paths." The agent now commits code without review in non-auth areas. A security incident later reveals the agent committed code with a SQL-injection vulnerability in a non-auth path. No tool changed. No authorization changed. A prompt condition changed behavior.

The practical test is simple: If a prompt change can alter what the product reads, writes, sends, refuses, escalates, logs, or approves, then a prompt change is equivalent to a code change. It belongs in the product-security supply chain with versioning, review, eval results, and a rollback path.

Prompt management must answer:

- Who can modify the prompt?
- How is a change reviewed?
- What evals must pass before the change ships?
- How is the old version preserved?
- Can the change be rolled back in production?
- Is there an audit log of prompt changes?

If the organization treats prompts as configuration that any engineer can change between deployments, the organization has lost visibility into a material source of product behavior.

Three Supply Chains, One Product

Traditional product-security supply chain oversight focused on software: source code, dependencies, containers, CI/CD workflows. A vulnerability in a dependency could compromise the entire product. Compromised builds could ship unsafe artifacts.

AI adds two more chains. They are not separate. They all converge on the same product.

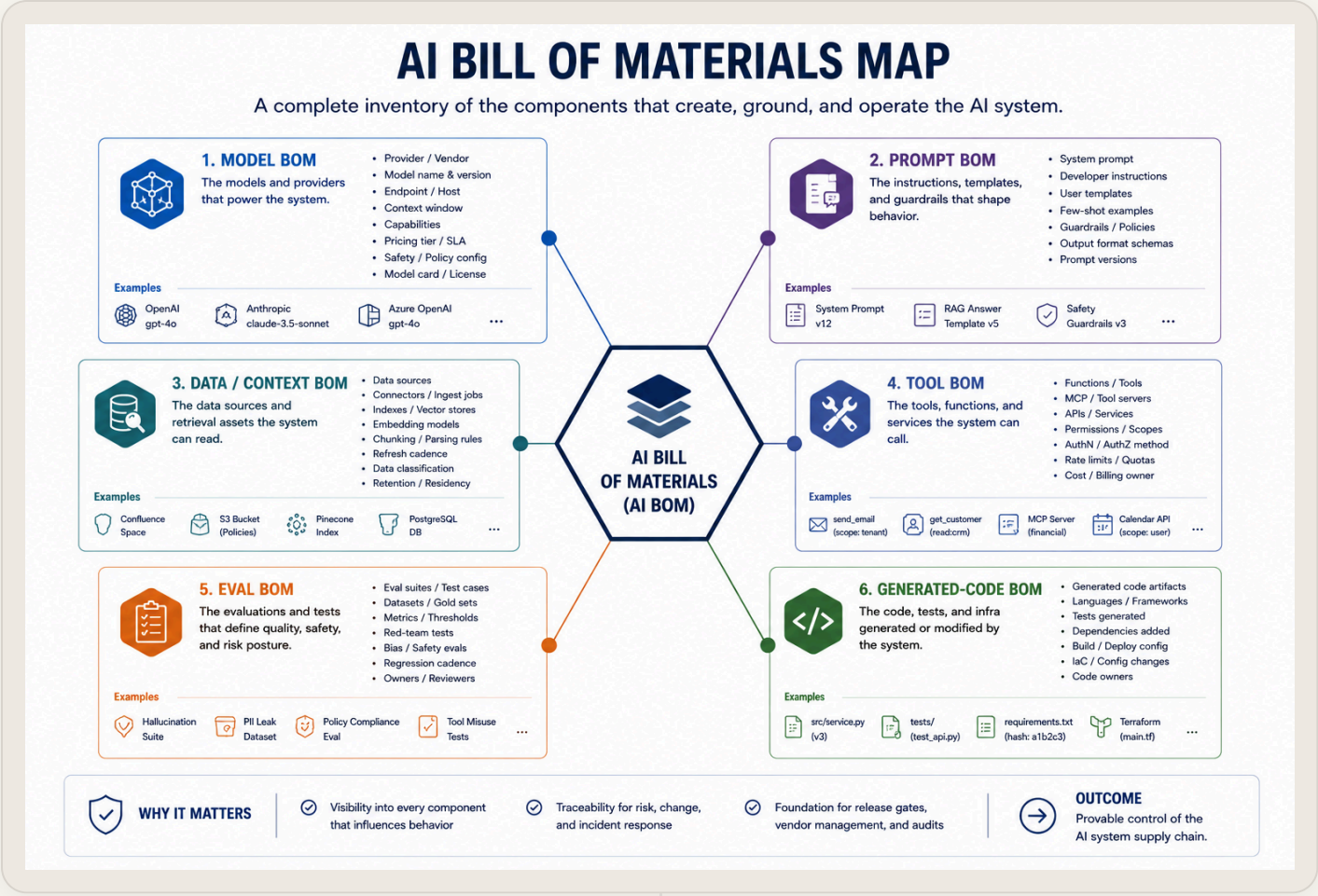

FIGURE 1: AN AI BILL OF MATERIALS MAPS MODEL, PROMPT, DATA/CONTEXT, TOOL, EVAL, AND GENERATED-CODE ARTIFACTS AS CONVERGING SUPPLY CHAINS REQUIRING VERSION, PROVENANCE, AND OWNERSHIP TRACKING.

Software chain: Source code, dependencies, containers, build pipelines, deployment workflows.

- Failure mode: vulnerable package, malicious code injection, unsafe generated code, compromised artifact
- Example: A dependency with a known vulnerability is vendored but never updated. An attacker exploits it to gain database access.

Model chain: Model weights, fine-tuned adapters, hosted endpoints, datasets, evaluation results.

- Failure mode: unverified provenance, unsafe or poisoned weights, undocumented training data, model behavior drift, untracked version
- Example: A team downloads a fine-tuned model from GitHub without verifying its source or how it was trained. The fine-tune silently degrades safety controls to improve performance. The model ships before anyone realizes.

Agent/Prompt chain: System prompts, prompt versions, tool servers, workflow definitions, generated code, evaluation assets.

- Failure mode: behavior drift due to prompt changes, overprivileged tool authority, unsafe generated code, invisible prompt modifications, evals that do not catch the actual risk
- Example: An engineer adjusts a prompt to be more concise. The change removes an instruction about checking tenant boundaries. The agent now retrieves across tenants. The change ships before anyone realizes it altered the security model.

Attackers do not care which chain creates the opening. They care whether the opening gives them execution, data, credentials, persistence, or influence over product behavior. An attack on the software chain might compromise a build and inject a backdoor. An attack on the model chain might poison weights or add a trigger that activates on certain inputs. An attack on the prompt chain might remove safety guardrails or escalation gates.

All three chains converge on the same product. All three require supply-chain oversight.

SolarWinds, Log4Shell, XZ Utils, PyPI poisoning, malicious GitHub Actions, and poisoned ML models demonstrate the pattern: an attacker does not need to compromise the core application. Compromising a supply chain artifact—a library, a plugin, a build script, a weight file, a prompt template—is enough to gain execution or influence. SLSA and SBOM frameworks exist because the threat is real and the surface area is large.

CISA's SBOM and AI inventory guidance points toward the same conclusion: AI-specific elements should be considered alongside general SBOM minimum elements. That means prompts, model versions, fine-tune metadata, tool inventories, and generated-code tracking are now supply-chain artifacts requiring the same rigor as code dependencies.

Prompts are now behavior-changing production artifacts. If a prompt change alters refusal behavior, tool selection, approval thresholds, escalation logic, output format, citation behavior, or logging expectation, it belongs in the supply chain. Treating prompts as harmless text is the AI-era version of treating build scripts as harmless glue.

AI-Generated Code Is a Contribution

AI-generated code should enter the same review path as human code, with additional scrutiny for the risks it tends to hide.

High-risk generated code should require:

- Human owner
- Review by the owning team
- Dependency and license checks
- Secret scanning
- Authorization and input-validation review
- Security tests for sensitive paths
- Clear rollback path

The rule is not "ban generated code." The rule is "do not let generated code bypass the controls required for the risk it touches."

Tool Servers Are Privileged Dependencies

Tool servers sit at the boundary between language and action. A model can suggest. A tool can read a database, send a message, open a pull request, execute a command, update a record, or trigger a workflow.

If a tool server is compromised, overprivileged, poorly logged, or allowed to shape context without validation, it can become the path from model output to product impact.

Tool-server review should address:

- Owner and purpose
- Authentication and token scopes
- Network reachability
- Input and output schemas
- Logging coverage
- Rate limits and tenant boundaries
- Error behavior and fallback

- Revocation and emergency stop

A tool server is a production dependency with action authority, not a harmless plugin.

The Supply Chain Standard

Every behavior-changing artifact needs provenance, version, review, evidence, and a rollback path appropriate to its risk. This includes:

- Source code and dependencies
- Container images
- Model artifacts
- Datasets and training data
- Prompt packages and system instructions
- Tool servers and plugins
- AI-generated code and contributions
- Eval assets and test suites

Detailed supply-chain mapping, artifact-type matrices, generated-code review templates, tool-server inventories, and rollback procedures – in Appendix G.

Sources

- NIST SSDF SP 800-218: https://csrc.nist.gov/pubs/sp/800/218/final
- SLSA: https://slsa.dev/
- Model Context Protocol: https://modelcontextprotocol.io/