AI PRODUCT SECURITY IN THE AGE OF MYTHOS · 2026

Chapter 07 · Threat Modeling Becomes Continuous

Standalone reading module for LMS delivery and required reading.

FORMAT	USE	SCOPE	AUDIENCE
Standalone PDF	Required reading	Single chapter	Learners

CHAPTER 07

Threat Modeling Becomes Continuous

108

PRODUCTS RED-TEAMED

Microsoft's AI Red Team reported 73 operations covering 108 products by September 2024. Threat modeling has to keep pace with system change, not meeting cadence.

MICROSOFT AI RED TEAM, 2025 GENERATIVE AI PRODUCTS

A threat model that does not alter the backlog is a conversation, not a control.

The Meeting That Changed Nothing

The product team schedules a threat-modeling session for their support agent. Seven people attend. The security architect walks through the use cases. The team maps data sources: Zendesk, Slack, Drive. Someone notes that the agent can send emails. Someone else notes that permissions are never rechecked. The team discusses whether the agent should have approval gates for sensitive actions. The conversation is thoughtful. They agree that external messaging is risky and that retrieval should happen with authorization. The meeting notes are thorough.

Two weeks later, the agent launches. External messaging is still automatic. Retrieval still happens before authorization. No eval was added. No approval gate was wired. No engineering ticket was filed. The notes sit in a Slack channel.

> ***Agreement is not risk reduction:*** *A threat model that does not alter the backlog is a conversation, not a control.*

The agent operates for three months. A customer escalation escalates. The agent sent an email to the wrong recipient using information retrieved across tenant boundaries. The incident response team reviews the threat model from that meeting. The team had identified both risks. The organization had discussed them. The product had shipped without fixing them.

The response was depressing because it was predictable: the threat model identified the risks, no one put them in the backlog, and risk became reality.

That is risk narration, not risk control.

Why AI Systems Require Continuous Threat Modeling

AI systems are not static. They change in ways traditional software does not.

A model provider releases a new version of Claude. The model is smarter and more capable. The token length doubles. The prompt injection resistance improves. Does the threat model change? Maybe. Does the ability to refuse harmful requests change? Possibly. Does the model's fit with your specific use case improve or degrade? No one knows until you test.

A development team adds a new tool. The agent used to summarize tickets. Now it can also draft email replies. The tool requires authentication but has production write access. The new capability crosses a line: from read-only to action. The old threat model is now incomplete.

A data team adds new documents to the RAG index. They are marked confidential but come from a folder with broad internal access. The indexing pipeline does not version permissions. Someone reclassifies a document later. The index does not update. Weeks later, the document is confidential again. But the stale chunk is still in the vector store.

An exception is granted. "Use the old model until Q3 while we migrate." Q3 arrives. The exception was not reviewed. No one closed it. The product continues on the old model with known issues.

A prompt engineer discovers that the system message drives behavior more than they realized. They adjust a single line. The new instruction causes the agent to escalate more aggressively to

humans. The change alters the threat model—now more actions go to approval—but no one updates the model.

A new contract with a customer adds a special exception. "Operate in a multi-tenant sandbox for this pilot." The architecture changes. Multi-tenant isolation now matters in a way it did not before. The threat model must reflect this.

Each of these changes is small. Each is a normal part of product development. But each changes the threat model. A model created once, stored as a diagram, and revisited once a year cannot keep up with this motion.

> ***Threat models in AI are not living diagrams:*** *They are feedback loops between product change and control change, or they are already stale.*

That was true before AI. Cloud infrastructure changes through IaC. APIs change through normal release cycles. SaaS workflows change through admin consoles. Feature flags change production behavior without a major deploy. CI/CD pipelines gain permissions. Tokens are added for convenience and forgotten after launch. AI systems accelerate the same drift: prompts change, model versions change, tools are added, retrieval sources expand, memory behavior shifts, and agents gain action classes one connector at a time.

Microsoft's red-team lessons from more than 100 generative AI products reinforce a critical point: AI systems amplify existing security risks and introduce new ones. The corollary is that static threat modeling becomes irrelevant fast. The threat model is not a document. It is a feedback loop between product change and control change.

Threat modeling must become a loop tied to product authority changes. When authority changes, the threat model changes. When the threat model changes, the controls should change.

Threat Modeling as Change Control

Continuous threat modeling is not endless meetings. It is embedding the threat-model question into the product change process.

Before a new tool is added, ask: what attack surfaces open? What actions can the agent now take? Before a new model version is deployed, ask: how does the new model change exploit risk or mitigation? Before a new data source is ingested, ask: what permissions matter, and how will stale ACLs be handled?

The question is not "is this risky?" Everything is risky. The question is "did the controls change with the product?"

An agent that could only summarize tickets gains ability to draft replies. Then it gains ability to send replies under a threshold. Then it gains escalation-note retrieval. Each change is small. But each change should trigger: What is the new threat surface? What is the new control requirement? Is there an eval? Is there an approval gate? Is there a log that proves the action happened?

Each change should leave a trail. The trail is the threat model becoming operational.

The Session Rule: Threats Must Produce Artifacts

Every threat-model session must end with one of four outcomes:

1. **A backlog item with owner and acceptance criteria**

The team identified a risk that requires engineering work. "Add approval gate for external messaging." "Implement ACL checks before context construction." "Add regression test for prompt injection." The ticket has a named owner who can make it happen, and acceptance criteria that prove it is done.

1. **A release blocker tied to a gate or eval**

The team identified a risk that blocks launch. "This agent cannot ship until the eval for external-message escalation passes." The gate exists in CI/CD. The eval has test cases. The gate will block the release if the eval fails.

1. **A risk exception with owner, reason, expiry, and review date**

The team identified a risk that the organization accepts for now. The support agent can send emails without approval during pilot because the customer explicitly requested speed. But the exception expires in 60 days. The CEO reviewed and signed off on the reason. On day 55, the system escalates this for executive review.

1. **A documented no-change decision with evidence**

The team discussed a potential risk and decided it is not a real concern. "We considered whether hostile customers could manipulate recommendations through crafted tickets, but the model's output is advisory only and requires human validation before action. The risk is within acceptable bounds." The evidence is written down. The next threat model revisit will check whether that evidence still holds.

If the outcome is "security will monitor," the team has not finished. Monitor what? Where is the log? Which alert? Which owner? Which threshold? Which review date? "Monitor" is not a control. Monitor is a hope.

This is the difference between continuous threat modeling and continuous conversation.

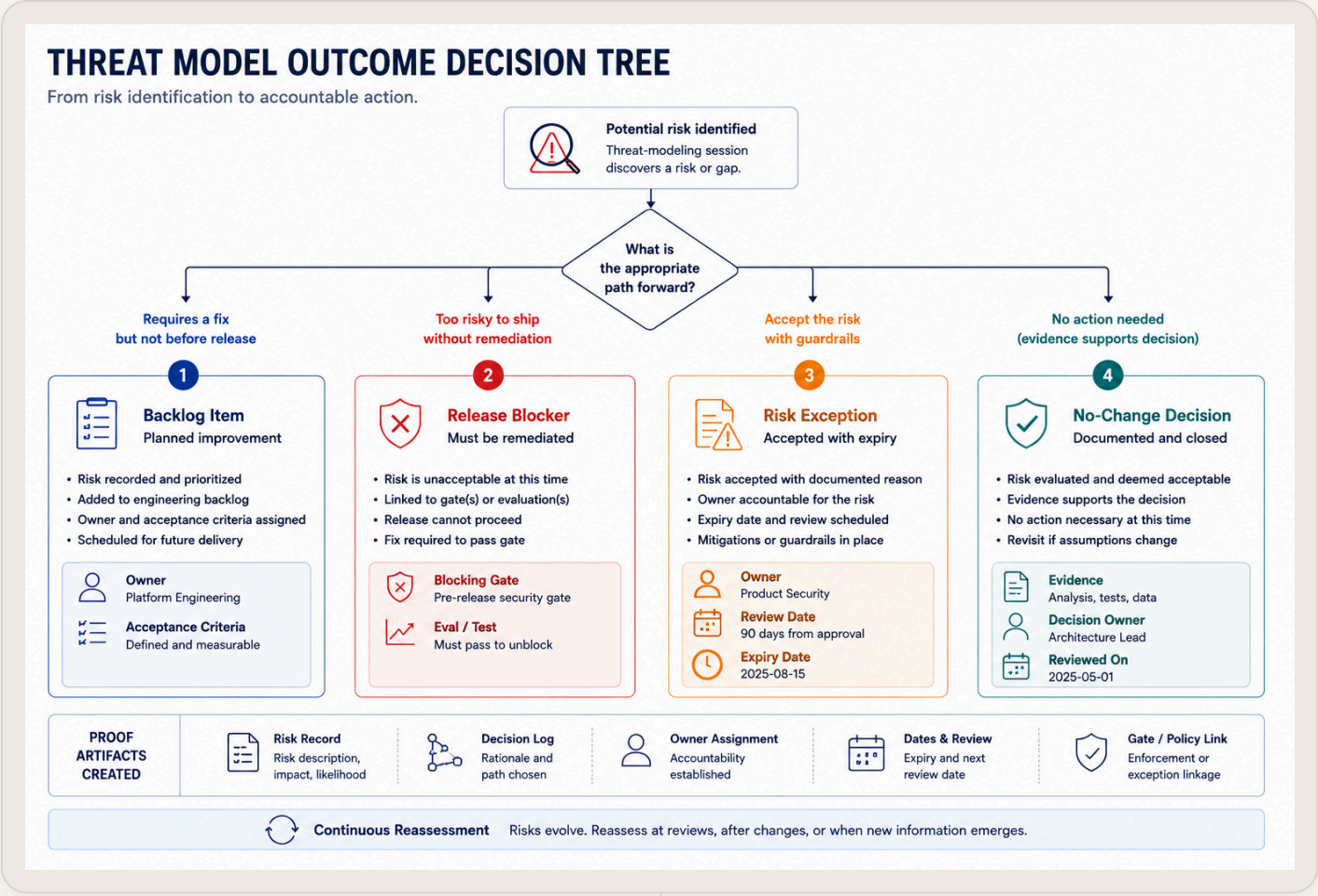

FIGURE 1: THREAT MODEL OUTCOME DECISION TREE: AFTER IDENTIFYING A RISK, THE TEAM CHOOSES ONE OF FOUR PATHS—BACKLOG ITEM, RELEASE BLOCKER, RISK EXCEPTION, OR DOCUMENTED NO-CHANGE DECISION—EACH PRODUCING PROOF ARTIFACTS.

The Backlog Test

A threat model has not changed risk until it changes at least one of these:

- **The backlog** – An engineering ticket was created with owner and due date.
- **The launch gate** – A blocking eval or approval was added to CI/CD.
- **The runtime policy** – A runtime decision is now enforced that was not before.
- **The approval flow** – An approval gate was added or modified.
- **The logging schema** – New events are now logged to enable incident response.
- **The exception register** – An exception was created or expired.

If none of those changed, the team may have improved shared understanding. It has not yet improved control. Understanding is a prerequisite. Artifacts are proof.

Where Continuous Threat Modeling Actually Breaks

Most teams understand the concept. Most teams do not operationalize it because the pressures are real:

Model upgrades happen outside the threat-model cycle. The ML team deploys Claude 3.5 Sonnet on Tuesday morning because it is faster and cheaper. The threat-modeling team does not convene until Friday. By Friday, the new model is in production. By next week, no one is sure what changed about the risk surface anymore.

Prompt changes are treated as configuration, not threat-surface changes. An engineer adjusts the system message to "be more concise"

or "escalate less frequently to save costs." These changes alter what the model does. They should trigger threat-model review. They do not. They are treated as normal development.

Release pressure bypasses the backlog. The threat model identifies a risk: "Agent can send external messages without approval." The backlog item sits for six months. Management wants the agent shipped. Someone suggests: "We can add the approval gate after launch." The agent ships. The gate never gets added because "the system is working fine in production."

Exceptions become permanent. An exception is granted: "Operate on old model version during migration. 60-day window." The clock stops because the migration keeps slipping. Two years later, the system is still running the old model. No one reviews the exception anymore.

Tool creep moves faster than threat modeling. Monday: "Can the agent update CRM status?" Engineer: sure, I'll add that connector. Wednesday: it is in production. Thursday: someone realizes this changed the threat model because now the agent can write data. By then it is too late to block. The team discusses adding eval, but there is no infrastructure yet. It goes on the backlog.

Continuous threat modeling fails not because the concept is wrong, but because **it competes with other pressures**. Model upgrades, feature requests, release deadlines, and exception renewals all have to slot into the threat-model cycle. When they do not, the model becomes a record of what was intended, not what is actually running.

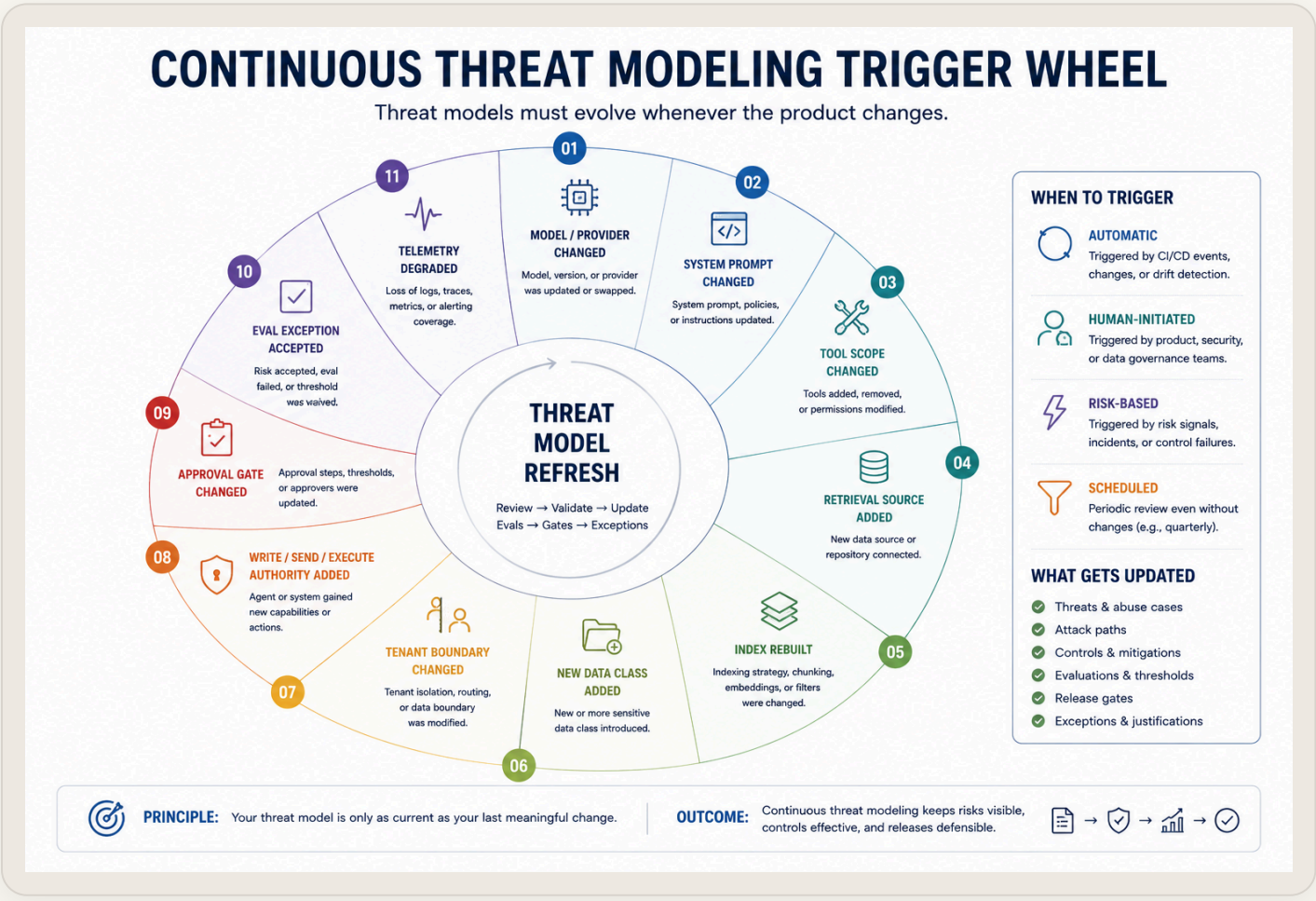

FIGURE 2: CONTINUOUS THREAT MODELING TRIGGERS SURROUND THE AI SYSTEM: MODEL CHANGES, PROMPT CHANGES, TOOL SCOPE CHANGES, DATA SOURCES, RETRIEVAL UPDATES, AUTHORITY ESCALATION, APPROVAL GATES, EVAL EXCEPTIONS, LOGGING CHANGES, AND EXTERNAL INCIDENTS ALL REQUIRE MODEL REFRESH AND DOWNSTREAM CONTROL UPDATES.

The antidote is not more process. It is enforcing at the control layer. A gate in CI/CD that requires threat-model review before a model version changes. A policy that prevents adding tools without eval. A release gate that blocks production unless the exception register is current. The threat model stays alive because the controls force it to stay alive.

The First Abuse Case Most Teams Rediscover

The first abuse case most teams will identify in AI threat modeling is not exotic. It is old confused-deputy logic wearing a language interface.

A user controls some input. The model sees the input and an internal tool. The model uses the tool to satisfy the user's request. The tool authority is broader than the user's authority. The user request influences the tool action. That is confused deputy.

It appeared in CGI scripts that trusted request parameters without checking permissions. It appears in APIs that escalate from user context to service context. It appears in prompt injection. The attack is not new. The surface is.

The next chapter explores this in detail.

AI Posture Reviews: Making Threat Modeling Repeatable

Continuous threat modeling works only if it is repeatable, standardized, and tied to operational decision gates.

An AI posture review is a structured threat-modeling engagement designed to be executed repeatedly—at intake, after product changes, or on a regular cadence—and to produce standardized artifacts that feed into the control plane.

A posture review should cover:

- **System Purpose and Scope** – What is the AI system? What does it do? Who uses it? What is its place in the product?
- **Model and Provider Details** – Which model? Which version? Which provider? Who approves model upgrades?
- **Data Classes and Sources** – What data does the system access? How is it classified? Who owns each source?
- **RAG and Context Retrieval** – Which documents or databases does the system retrieve from? How are permissions enforced? Is retrieval eligible before context assembly?
- **Tool and Action Permissions** – Which tools can the system invoke? Are there approval gates? Which actions require human sign-off?
- **Identity and Token Boundaries** – What identity does the system use? Are tokens scoped? Can the system escalate privileges?
- **Prompt-Injection Exposure** – Have direct injection vectors been tested? Have indirect injection paths (via retrieved content) been assessed?
- **Output Safety and Filtering** – Are there guardrails? Are they tested? What data should the model never return?
- **Logging and Evidence Requirements** – What events must be logged? Can incident responders reconstruct what happened?
- **Incident Response Readiness** – If this system is breached or misused, can the organization detect it and respond?
- **Regulatory and Compliance Scope** – What regulations apply? What audit evidence is required?
- **Risk Assessment and Controls** – What are the top 3-5 risks? What controls mitigate them? Which risks are accepted, and on what timeline?

- **Governance Signoff** – Who owns this system? Who approved the risk assessment? When is the next review?

A posture review produces three key artifacts:

1. **Risk Checklist** – Structured assessment of threat vectors and control status, mapped to NIST AI RMF, OWASP LLM Top 10, and MITRE ATLAS where applicable.
2. **Authority Graph** – Visual or documented model of data access, tool permissions, approval paths, and identity boundaries.
3. **Control Roadmap** – Backlog items, blocked risks, exceptions with expiry, and evidence requirements that will prove control.

Organizations that operationalize posture reviews—making them part of the AI intake process, requiring them before major changes, and scheduling them annually—turn threat modeling from a one-time engagement into a repeatable operational control. The threat model stays alive because the review makes it a condition of continued operation.

Detailed threat-model triggers, backlog translation templates, high-risk acceptance criteria, failure-mode patterns, and posture-review templates – in Appendix C.

Sources

- Microsoft, Lessons from Red Teaming 100 Generative AI Products: https://openreview.net/pdf?id=auiAIKsJXg
- NIST AI RMF: https://www.nist.gov/itl/ai-risk-management-framework
- MITRE ATLAS: https://atlas.mitre.org/
- OWASP Top 10 for LLM Applications 2025: https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025