How to threat model agentic AI systems.

Agentic AI threat modeling is the structured process of finding how an AI agent system can be attacked or misused before it ships. Start from the business problem, inventory business and AI assets, draw data flow diagrams for every agent, tool and data store, and mark trust boundaries between users, agents, tools, databases and third parties. Enumerate threats with STRIDE, MITRE ATLAS, the OWASP LLM and agentic Top 10 lists, CVE and CWE, then score likelihood, impact, business and compliance risk. Update the model whenever the architecture or threat landscape changes.

HOW IT WORKS

01 Step 1: Start with the business problem and the right people

Threats only make sense against what the system is for. Before drawing anything, write down in plain language what the agent does, for whom, and what a bad day looks like. Our running example: a customer-support agent that reads tickets, looks up orders, answers customers and can issue refunds.

Get the right people in the room, because no single team sees the whole system:

- Product and business owners know which actions are expensive, irreversible or regulated, and what the acceptable loss is.
- Engineers and ML engineers know the real architecture: prompts, models, tools, memory and where data actually goes.
- Security, privacy, legal and compliance know the threats, the obligations and the audit requirements.
- Operations and support know how the system will really be used, and abused, day to day.

The output of this step is a short statement of purpose, the critical actions (for the refund agent: issuing money and exposing customer data), and the business impact if each goes wrong.

02 Step 2: Build an asset inventory for the AI system

List everything worth protecting or worth attacking, in two groups.

SOURCES

[1] MITRE ATLAS

[2] OWASP Top 10 for LLM Applications (2025)

[3] OWASP Top 10 for Agentic Applications (2026)

[4] OWASP Threat Modeling Process

Business assets are what the organisation cares about directly:

- Customer personal data and order history
- Money movement: refunds, credits, payment details
- Business records, internal knowledge and policies
- Reputation and regulatory standing

AI assets are the parts that make the agent work, and each can be targeted:

- Models and model configuration, including the provider and version
- System prompts and agent instructions (see system prompt extraction)
- Tools and their credentials, including MCP servers
- Agent memory and conversation history (see agent memory poisoning)
- Retrieval stores, vector databases and their source documents (see RAG poisoning)
- Training and fine-tuning data, evaluation sets and logs
- Agent identities, API keys and OAuth tokens (see AI agent identity)

For each asset, record its owner, its sensitivity and where it lives. You will need all three when you score risk.

03 Step 3: Draw data flow diagrams for every agent

A data flow diagram for an agentic system has the usual elements (external entities, processes, data stores and flows) plus the parts that make agents different. For a multi-agent system, draw:

- Every agent as its own process, with its model, its system prompt and the tools it can call. In our example: a triage agent that reads tickets, and a refund agent that can call the payments API.
- Every tool call as a flow, including what goes in (arguments the model chose) and what comes back (output that re-enters the model's context).

- Agent-to-agent messages. When the triage agent hands a task to the refund agent, that message is untrusted input to the refund agent, even though both are yours.
- Memory and retrieval as data stores that agents both read and write.
- Every source of natural language the agent reads: user chat, ticket bodies, emails, attachments, web pages, tool results.
- Third-party services: the model provider, SaaS APIs and any external MCP servers.

The question to ask at every arrow: could an attacker control or influence the text that travels along this flow?

04 Step 4: Define trust boundaries

A trust boundary is any line where data or control crosses between parts with different levels of trust. In agentic systems the important ones are:

- User to agent: everything a user types is untrusted, including authenticated users.
- Content to agent: documents, emails, tickets and web pages the agent reads are untrusted, whoever wrote them.
- Agent to tool: the model's choice of tool and arguments is untrusted output; the tool must enforce its own authorization.
- Tool to agent: tool results flow back into the model's context and can carry instructions.
- Agent to agent: one agent's output is another agent's untrusted input.
- Agent to database: queries built from model output must be parameterised and limited to the agent's own scope.
- Your system to third parties: the model provider, SaaS APIs and external MCP servers each sit behind their own boundary, with their own data handling.

The most useful rule this step produces: the model is never the security control. Every boundary needs enforcement in code, an identity check, a scoped credential or a human approval, not an instruction in the prompt.

05 Step 5: Put hard controls on critical actions

Walk every action the agent can take and ask what happens if the agent is fully manipulated into taking it. For the refund agent, this is where the threat model earns its keep:

- Refunds need approval controls in code, not in the prompt. Set hard limits (per refund, per customer, per day), require the refund to match an existing order and amount, and send anything above the limit to a human queue. A prompt that says "never refund more than $100" is a suggestion an attacker can talk the model out of.
- Critical financial transactions need human approval. Payments, large refunds, credits to new accounts and changes to bank details should require a person to approve the concrete action (amount, recipient, order), shown in full, not a summary written by the agent.
- Separate read from write. The triage agent should not hold payment credentials at all. Give each agent only the tools its task needs.
- Make actions reversible where possible, and add a kill switch that disables a tool or agent everywhere at once.

06 Step 6: Design logging and monitoring carefully

Agent incidents are hard to investigate without the right records, and easy to make worse with the wrong ones. Design logging as part of the threat model:

- Log the decision trail: the user request, the agent and user identities, each tool call with its arguments and result, approvals and denials, and the model and prompt version in use.
- Do not log sensitive PII in clear text. Conversations and tool results are full of names, addresses, order details and sometimes card or health data. Redact or tokenise personal data before it reaches logs, keep full transcripts only where there is a real need, restrict who can read them, and set retention limits. Logs are a favourite exfiltration target and often fall under the same privacy obligations as the source data.

- Never log secrets. Tokens and keys that appear in prompts or tool output must be scrubbed.
- Monitor for behaviour, not just errors: refund volumes and amounts per customer, tool calls outside normal patterns, repeated denials, and spikes in cost that suggest denial of wallet.

07 Step 7: Enumerate threats

With assets, flows and boundaries mapped, enumerate threats systematically. Combine frameworks, because each sees different things:

- STRIDE gives the classic categories at each element and boundary: Spoofing (an attacker impersonates a user or another agent), Tampering (poisoned memory or retrieval data), Repudiation (no record of which agent issued a refund), Information disclosure (the agent leaks another customer's order), Denial of service (looping agents, runaway token cost) and Elevation of privilege (prompt injection makes the triage agent trigger a refund).
- MITRE ATLAS catalogues real adversary tactics and techniques against AI systems, from reconnaissance to exfiltration. Use it to make sure you have considered how attacks on models actually happen.
- The OWASP Top 10 for LLM Applications covers model-level risks such as prompt injection, sensitive information disclosure and improper output handling. See the OWASP LLM Top 10.
- The OWASP Top 10 for Agentic Applications adds agent-specific risks such as agent goal hijacking, tool misuse, identity and privilege abuse, and memory and context poisoning.
- CVE and CWE keep the traditional stack in scope. The agent framework, the vector database, the web layer and every library have their own known vulnerabilities (CVE) and weakness classes (CWE).

For the refund agent, a realistic chain looks like this: a customer embeds instructions in a support ticket, the triage agent passes them to the refund agent as part of the task, and the refund agent issues a refund to an order the customer does not own. That single scenario touches spoofing, elevation of privilege, prompt injection and excessive agency at once.

08 Traditional AppSec controls still matter

Most real agent breaches still go through ordinary weaknesses. The agent sits on a web application, calls APIs and queries databases, so the usual controls remain essential:

- Authentication and object-level authorization on every API the agent calls, so a manipulated agent still cannot read or change another customer's data.
- Parameterised queries and output encoding, because model output is untrusted input to whatever consumes it. See LLM output validation.
- Secrets management, rate limiting, dependency scanning and patching.
- Network egress controls and sandboxing for any tool that executes code. See AI agent sandboxing.

A useful test of a design: if the model were replaced by an attacker typing tool calls directly, what could they do? The answer should be "only what this user is already allowed to do".

09 Assessing third-party agentic AI systems

When the agent, the platform or a key tool comes from a vendor, the threat model has to cover what you cannot see. Ask for documentation, and treat gaps as risk:

- An architecture overview and data flow diagram, including which models and sub-processors are used and where data is processed and stored.
- Data handling terms: retention, whether your data is used for training, deletion on request and data residency.

- How the agent is authorised: identities, scopes, how credentials are stored, and whether it can act with more privilege than the user.
- Controls for human approval, action limits and kill switches.
- Logging and audit capabilities, and what you can export.
- Security testing evidence: recent penetration tests and AI red-team results covering prompt injection and tool misuse, plus how findings were fixed.
- Certifications and reports such as SOC 2 or ISO 27001, and incident notification commitments.
- How model and prompt changes are versioned and communicated, since a silent update can change behaviour.

10 Step 8: Score and prioritise risk

Rate every threat on four dimensions:

- Likelihood: how reachable is it? Untrusted text reaching an agent with write tools is highly likely to be attempted.
- Technical impact: what can the attacker do or reach if it works?
- Business risk: financial loss, fraud, operational disruption and reputational damage.
- Compliance risk: exposure of personal data, payment card data or regulated information, and obligations under privacy and sector rules.

Prioritise threats that are both likely and high impact, then pick controls that reduce impact even when the model is fooled: scoped credentials, hard limits, approvals and isolation. Record accepted risks with an owner and a review date.

11 Keep the threat model alive

An agentic threat model goes stale faster than most. Update it whenever any of these change:

- A new tool, MCP server, data source or agent is added
- The model, provider or system prompt changes

- Permissions, approval limits or workflows change
- New attack techniques appear, such as new entries in MITRE ATLAS or the OWASP lists
- An incident, near miss or red-team finding shows something the model missed

Validate the model by testing it: an AI red team exercise against the real system shows whether the controls you designed hold when someone actively tries to break them.

Test your AI application before an attacker does.
securelayer7.net/learn/ai-security/agentic-ai-threat-modeling

Open online