



RESEARCH REPORT

Threat Modelling Autonomous AI Agents

A practical threat model for systems where a language model can take actions — tool access, prompt injection, credential scope, and the audit trail you will wish you had.

PUBLISHED

August 19, 2026

READING TIME

15 minutes

AUTHOR

SCIAT AFRICA Research

ABSTRACT

An AI agent that can call tools is a new kind of privileged user: it holds credentials, takes actions with side effects, and decides what to do next from text it did not write. This report sets out a threat model for agentic systems and the controls that constrain them – written for teams shipping agents into production rather than evaluating them in a lab.

Topics: AI Security · Application Security · Threat Modelling

WHAT TO TAKE AWAY

- Treat every token an agent consumes as untrusted input, including tool output, retrieved documents and page content.
- The blast radius of an agent is the union of its tool permissions – scope credentials to the task, not to the user.
- Indirect prompt injection via retrieved content is the dominant attack path and cannot be fixed by prompt wording alone.
- Actions with real-world side effects need a confirmation boundary that the model cannot cross on its own.
- Agent runs need an immutable audit trail at the level of tool calls and arguments, or incidents are uninvestigable.

CONTENTS

- 01 What is actually new here
- 02 Indirect prompt injection is the main event
- 03 Credential scope decides the blast radius
- 04 Side effects need a boundary the model cannot cross
- 05 Memory, persistence and cross-session contamination
- 06 Making agent incidents investigable
- 07 How we test agentic systems

01 What is actually new here

Language models have been in production for years in a mostly inert configuration: text in, text out, a human deciding what to do with the result. Agentic systems remove the human from the loop between decision and action. The model reads, decides, calls a tool, reads the result, and decides again. That loop is the new attack surface, and it has no direct analogue in conventional application security.

The useful mental model is that the agent is a privileged user whose instructions arrive from an untrusted channel. Everything that follows is an application of that one idea. The OWASP Top 10 for Large Language Model Applications is a good public reference point and names prompt injection as the leading risk; this report is about what to build in response.

02 Indirect prompt injection is the main event

Direct injection — a user typing instructions that override the system prompt — is well known and comparatively easy to reason about. The harder case is indirect: the agent retrieves a document, a web page, an email, a code comment, or the output of a tool, and that content contains instructions. The model has no reliable mechanism for distinguishing data from directive, because to the model both are tokens in the same context window.

Mitigations that operate inside the prompt — telling the model to ignore instructions in retrieved content — reduce success rates without eliminating them, and should not be treated as a control boundary. The controls that hold are architectural: limit what the agent can do, so that a successful injection has nowhere useful to go.

- Mark provenance on every piece of context, and keep retrieved content out of the system-instruction position.
- Deny by default on tool access; grant per task, not per session.
- Require human confirmation for irreversible or outward-facing actions, enforced outside the model.
- Never place a long-lived secret in the context window; give the agent a broker, not a credential.

03 Credential scope decides the blast radius

The most consequential finding in our assessments of agentic systems is almost always the same: the agent runs with the permissions of the human it serves, or with a service account provisioned broadly for convenience. A single successful injection then reaches everything that human or account reaches.

The alternative is narrow, short-lived, task-scoped credentials issued by a broker that the agent calls, with the broker — not the model — enforcing which operations are permitted in which context. This is more work to build, and it is the difference between an incident that is contained to one task and one that is contained to nothing.

04 Side effects need a boundary the model cannot cross

Reading is recoverable. Sending an email, moving money, deleting a record, publishing a page, opening a pull request, messaging a customer — these are not. The distinction should be structural, not a matter of prompt instruction: the agent proposes, and a mechanism outside the model decides whether the proposal executes.

That mechanism can be a human approval step, a policy engine, a rate limit, a staging environment, or an allowlist of recipients. What it cannot be is the model agreeing not to do it. Any control whose enforcement point is inside the context window is advisory.

05 Memory, persistence and cross-session contamination

Agents that persist memory across sessions introduce a storage attack: content injected once is retrieved later, in a different and possibly more privileged context. We have seen this pattern produce the longest-lived issues, because the injection is no longer visible in the conversation that triggers it.

Treat written memory as untrusted on read, not only on write. Record where each memory came from, scope memories to the context that created them, and expire them. Where an agent summarises its own history, remember that the summary is model output derived from untrusted input and carries the same taint.

06 Making agent incidents investigable

When something goes wrong, the question is always the same: what did the agent do, in what order, with which arguments, on whose behalf, and what did it read immediately before. Systems that log only the final response cannot answer it.

The audit trail needs the full tool-call sequence with arguments and results, the provenance of retrieved context, the identity the action was taken under, and the model and version in use. It needs to be written somewhere the agent itself cannot modify. This is cheap to build on day one and close to impossible to reconstruct after an incident.

07 How we test agentic systems

Our assessments combine a design review against this threat model with adversarial testing of the running system: indirect injection through each retrieval channel, attempts to escalate from read-only to write tools, extraction of system instructions and credentials from context, chaining of individually-permitted tool calls into an unintended outcome, and review of the audit trail's completeness against what we were able to do.

The deliverable is a prioritised findings report with reproduction steps, plus the architectural changes that close the class of issue rather than the instance.

About this publication

This document is published by SCIAT AFRICA, a cybersecurity and digital forensics agency based in Nairobi, Kenya. It is free to read, print and circulate inside your organisation. The web edition, which carries the same content, is at <https://sciatafrica.com/research/ai-agent-threat-model>.

It is general guidance, not legal advice and not a substitute for an assessment of your own environment. Where it refers to a law, regulation or standard, read the instrument itself before relying on it – and talk to us if you want help applying it to your estate.

SCIAT AFRICA

info@sciatafrica.com · +254 794 946633 · <https://sciatafrica.com>