Turn "how do you secure your AI agents?" from a deal-stopper into a deal-closer.

Independent proof your agents cannot access, execute, or leak what they should not - the answer your biggest customers' security teams now demand.

Authorized-only Human-verified Firm-attributed

Contact: airiskmanagement.ca/contact · info@airiskmanagement.ca

Executive summary

The risk, the proof, and a free look from the outside

The shift Your AI agents now access data, execute actions, and take steps for users - and your biggest customers' security reviews now ask how you secure that. A weak answer stalls the deal.

The proof We already find exactly this class of flaw in the AI and agent tooling your stack is built on: cross-tenant data access, tool command-boundary abuse, and SSRF or exfiltration.

The free look From public data only (no access needed) we show you what your AI stack looks like from the outside - where you are exposed across access, execute, and leak - then a 20-minute call on what to fix first.

See your public footprint

Public sources only. Human-verified deeper work begins only under a signed rules-of-engagement.

The security boundary just moved

Your agents, copilots, and MCP servers now read your data, call your tools, and take actions for users. The security boundary moved from your app to what your agent is allowed to do - and your enterprise customers have started asking about it in their security reviews.

Three questions your board should be able to answer about your AI

Cross-customer breach

Access

Can one user, tenant, or role reach another's data through our agents?

Record or account takeover

Execute

Can our agents or tools be driven to take unauthorized actions?

Exfiltration, internal pivot

Leak

Can our AI be turned into an exfiltration or SSRF path?

If you cannot answer these three about your AI, neither can your customers' security teams.

Access

One customer read another's data - through the agent

What we found

In a widely-used open-source RAG/agent platform, an agent tool returned another tenant's documents - the ownership check lived one layer above the data service, so the agent path skipped it. We see this pattern repeatedly: the human UI enforces the check, the agent tool does not.

What it would mean for you

One customer reads another's contracts, attachments, or PII - a cross-tenant breach with notification duties.

Would your current testing catch it?

Usually not. Only the agent path was missing the check.

Execute

One user overwrote another's records - and locked them out

What we found

In a self-hosted data-agent, the only write path had no ownership check - one user could overwrite another's records and lock them out. In a separate agent server, a tool accepted an unconstrained parameter that pushed it past its intended boundary.

What it would mean for you

Silent data tampering, record or account takeover, or an agent coaxed into an action it should refuse.

Would your current testing catch it?

Rarely. This is agent and tool command-boundary logic, not a scanner signature.

Leak

The AI became a path into our internal network

What we found

An AI ingestion path followed an attacker-controlled URL (SSRF from inside the integration), and a verbose error exposed internal service hostnames.

What it would mean for you

Your AI becomes a pivot into internal systems, or a slow exfiltration channel.

Would your current testing catch it?

Often not. The request comes from a trusted internal service, so it looks legitimate.

Your current testing has an AI-shaped blind spot

Penetration test - covers your app surface

Scanners - cover known CVEs

Bug bounty - covers what researchers pick

The agent layer: access, execute, leak - nobody is testing this

The gap is exactly where the next class of incidents will come from.

This is the class of flaw we find - for real

Open-source RAG platform

Access

Cross-tenant document read

Self-hosted data agent

Access Execute

Record takeover, missing ownership check

MCP server

Execute

Tool driven past its intended boundary

CI/CD automation plugin

Execute

Command execution via the plugin

The same flaw classes we find in the ecosystem - and here is how we spot them from the outside, on your public footprint, before we touch anything.

SAMPLE - populated per prospect

What we can see about [your-domain.com] - from the outside

All from public sources: certificate logs, Shodan/Censys, public code and DNS. We never connect to your systems.

Stack we inferred (from public signals): [agent framework] · [vector/RAG store] · [cloud] · [model provider]

Access	[exposed staging subdomain / admin login found in certificate-transparency logs]
Execute	[internal tool or agent endpoint referenced in a public code repo]
Leak	[API-key pattern in a public commit, or a verbose public service banner]

Full findings, no score, no grade. Book a call and we walk through what matters and what to fix first.

The Review: proof you can hand your customers

A verified risk register across access, execute, and leak; reproduction-safe evidence; developer-ready fixes; and a retest to closure - packaged to pass your customers' security reviews.

Every finding is tagged to OWASP LLM Top 10 and MITRE ATLAS and mapped to your SOC 2 and NIST AI RMF controls - audit-ready evidence your assessor uses, while your auditor still issues the opinion. A compliance platform confirms a control exists; we prove your agents cannot be abused. Authorized-only, human-verified, firm-attributed.

10 business days

Signed rules-of-engagement

How it works

1 · Public-exposure snapshot — Free

Free, from public data - no access to your systems.

2 · Threat-model call — Free

Free, 60 minutes: what it means and the single top fix.

3 · Diagnostic or Review — Paid

Fixed-fee, under a signed ROE: verified findings, evidence, remediation, retest.

4 · Release Security Gate — Ongoing

Ongoing retainer: security sign-off on each release.

Book a free threat-model call