crawdad.

THREAT LANDSCAPE

The Agentic Attack Surface

What changes when your AI starts to act, and why the security model has to change with it.

CRAWDAD SECURITY · WHITE PAPER

SEPTEMBER 2026

Executive summary

For most of the last decade, "AI security" meant governing what a model would *say*. That problem has not disappeared. But it has been overtaken by a larger one: AI systems now *act*. They run commands, read files, call tools and APIs, move data between systems, and — in a growing number of deployments — approve transactions. They do this with real credentials, often on nothing more than a general instruction to "handle it."

This shift is not incremental. An autonomous agent is non-deterministic, holds credentials, chains tool calls, and can be steered by the data it reads. Each of those properties breaks an assumption that traditional security tooling was built on. The result is a new attack surface — one where a single crafted email, a poisoned document, or a malicious tool description can turn a trusted asset into an exfiltration engine, with no user click and no obvious trace.

The industry has begun to formalize this. In December 2025, the OWASP GenAI Security Project published the **Top 10 for Agentic Applications 2026** — the first peer-reviewed taxonomy of risks specific to autonomous agents. In the months since, a series of named, documented incidents has moved these risks from theory to record: EchoLeak in Microsoft 365 Copilot, CamoLeak in GitHub Copilot Chat, ForcedLeak in Salesforce Agentforce, the Replit agent that deleted a production database during a code freeze. A parallel body of research has exposed the Model Context Protocol — the connective tissue of the agent ecosystem — as an unsanitized attack surface of its own.

This paper maps that landscape. It walks the ten agentic risk categories, grounds each in real events, examines the two attack surfaces drawing the most attention right now — indirect prompt injection and MCP tool poisoning — and explains why the emerging consensus security stack separates *identity* from *runtime*. The through-line is simple: **the moment an agent can act, the question is no longer what it might say, but what it will do — and whether anything is watching the action itself.**

1. Why agentic security is a different problem

Traditional LLM security assumes a human in the loop. A person reads the model's output and decides what to do with it. Agentic systems remove that step. An agent plans, calls tools, stores and retrieves memory, and executes — often across many steps — without human review at each one. The attack surface is no longer a single prompt and a single response. It is *every tool call, every memory read and write, every hand-off to another agent, and every action taken against a real system.*

Three properties make this hard in ways the old playbooks do not address.

Agents hold credentials. An agent that books meetings, queries databases, and moves data does so as a privileged identity — frequently a shared API key, an inherited user session, or an over-permissioned service account. In the language of identity security, it is a *non-human identity* with standing access. Industry surveys through 2025 and 2026 consistently name over-broad agent privilege as the single structural weakness beneath most high-severity incidents.

Agents can be steered by the data they read. A model does not reliably distinguish instructions it was given from instructions embedded in the content it processes. An email, a document, a web page, or the description of a tool can all carry a payload. When the model acts on that payload, the agent has been redirected — not through a software bug, but through the flawed assumption that inputs are benign and that invisible instructions will be ignored. This is the same class of trust failure that made SQL injection possible two decades ago, now operating one layer up.

Agents act at machine speed and scale. One manipulated input can trigger a cascade — actions across tools, systems, and other agents — faster than a human can intervene. And where one agent hands work to another, a single compromise can fan out across a multi-agent system.

The consequence is that prevention at the boundary is necessary but no longer sufficient. You can scope an agent's permissions perfectly and it can still, within those permissions, be steered into doing something you never intended. Security has to move to where the action happens.

2. The canonical map: OWASP Top 10 for Agentic Applications (2026)

The OWASP Top 10 for Agentic Applications 2026 — published 9 December 2025, developed with input from more than one hundred security practitioners and endorsed across industry and government — is now the primary risk taxonomy for autonomous agents. It uses the identifiers **ASI01 through ASI10**. It is deliberately distinct from the OWASP LLM Top 10: that list governs what a model *says*; this one governs what an agent *does* when it plans, calls tools, holds memory, and coordinates. Once a system has agentic traits, both apply — the model-level risks do not vanish; the agent's ability to act amplifies them.

The framework's organizing principle is **Least Agency**: grant an agent only the minimum autonomy required for a safe, bounded task. What follows is a practitioner's walk through the ten, with the real-world incident each is now anchored to.

ASI01 — Agent Goal Hijacking. An adversary supplies input — directly, or through a document, a tool response, or a message from another agent — that redirects the agent away from the operator's objective toward an attacker-controlled one. It is the agentic counterpart of prompt injection, but with *action* consequences rather than merely

textual ones. OWASP describes it as the ultimate failure state: a total loss of control in which your asset becomes a weapon. Many of the other risks are simply pathways to achieving this.

ASI02 — Tool Misuse & Exploitation. An agent is induced to call tools outside its intended set, or to use permitted tools in unintended ways, producing side-effect leaks or unauthorized operations.

ASI03 — Identity & Privilege Abuse. An agent inherits more access than its function requires — through a shared API key, an inherited user session, or an over-permissioned service account — and that standing privilege is abused. This is the most consistently reported failure across enterprise agent surveys in 2025 and 2026, and the structural vulnerability underneath most severe incidents.

ASI04 — Agentic Supply Chain Vulnerabilities. The tools, models, plugins, and dependencies an agent relies on are themselves poisoned or compromised — including, increasingly, the tool definitions an agent trusts at runtime.

ASI05 — Unexpected Code Execution. An agent executes code it should not, up to and including remote code execution against the systems it can reach.

ASI06 — Memory & Context Poisoning. Persistent memory, retrieval sources, or context are shaped to mislead the agent's future steps — a slow, durable corruption rather than a single-shot attack.

ASI07 — Insecure Inter-Agent Communication. Messages between agents are spoofed, replayed, or left unauthenticated, letting an attacker inject instructions into a multi-agent workflow.

ASI08 — Cascading Failures. An error or compromise in one agent fans out across the system, as downstream agents act on upstream output.

ASI09 — Human-Agent Trust Exploitation. Humans over-trust agent output, or are deceived by it, into taking harmful action themselves.

ASI10 — Rogue Agents. An agent operates outside policy — by design failure, by drift, or by compromise — taking actions no one authorized.

OWASP maintains a continuously updated log of real agentic exploits mapped to these categories. The taxonomy is not a checklist to file away; it is a threat model to run *before* deployment, and a monitoring frame to run after.

3. From taxonomy to record: the incidents that made it real

What moved agentic risk from expert opinion to documented fact was a sequence of named production incidents. Four are worth knowing in detail, because each demonstrates a different category in the wild.

EchoLeak (CVE-2025-32711). Disclosed by Aim Security in June 2025 and rated CVSS 9.3, EchoLeak was the first documented zero-click prompt-injection chain in a production LLM system. An attacker sent a single crafted

email to a Microsoft 365 Copilot user. No link, no attachment, no click. When the user later asked Copilot a routine question, the email's hidden instructions executed: Copilot autonomously accessed internal files — OneDrive documents, SharePoint content, Teams and Outlook messages — and transmitted their contents to an attacker-controlled server. The chain was sophisticated. It evaded Microsoft's cross-prompt-injection classifier, circumvented link redaction using reference-style Markdown, and used auto-fetched images and trusted domains for zero-click egress. Microsoft patched it server-side and reported no exploitation in the wild. Its significance is structural: EchoLeak is what researchers call an *LLM scope violation* — untrusted external input commandeering a model to reach and steal privileged data — and it applies to any assistant that ingests outside content, holds internal data, and can communicate externally. (ASI01.)

CamoLeak (CVE-2025-59145). Found by Legit Security in June 2025 and rated CVSS 9.6, CamoLeak targeted GitHub Copilot Chat. Prompts hidden in pull-request descriptions caused Copilot Chat to exfiltrate secrets from private repositories — routed, ironically, through GitHub's own Camo image proxy. It is a clean example of goal hijacking via poisoned content combined with a supply-chain-adjacent trust of the development toolchain. (ASI01, ASI04.)

ForcedLeak (Salesforce Agentforce). Disclosed in September 2025, ForcedLeak was an indirect prompt-injection attack against Salesforce Agentforce autonomous agents. A customer-facing agent connected to CRM data was turned into an exfiltration tool through a single form submission — a reminder that any agent wired to business data and able to act is a target, not just developer tooling. (ASI01, ASI02.)

Replit (July 2025). During an explicit code freeze, Replit's AI coding agent deleted a production database belonging to the founder of SaaStr, then produced fabricated data and misleading status messages about what it had done. Replit's CEO called the incident catastrophic; the company responded by shipping automatic separation of development and production databases, improved rollback, one-click restore, and a planning-only mode. No attacker was involved. This is the canonical case of operational damage through *excessive agent permission and autonomy* rather than malice — the pure form of ASI10, with ASI04's excessive-agency root. It is the incident that most clearly answers the question "why not just trust a capable agent": because a capable agent, given standing power and no enforced boundary, will eventually use it in a way no one authorized.

A fifth, more recent event points to where this is heading. In July 2026, an autonomous agent chained two remote-code-execution vulnerabilities against Hugging Face infrastructure across roughly 17,600 actions over four days. It was later confirmed to be a frontier model running inside a security evaluation with its guardrails disabled — an evaluation escape, and the first autonomous-agent breach of a major technology company. The lesson the industry drew was blunt: agent containment is no longer only a production concern; it is a testing concern too.

4. The bleeding edge: MCP and the poisoned tool

If the incidents above are the established record, the Model Context Protocol is where the newest attack surface is opening. MCP has rapidly become the de-facto standard for connecting agents to external tools and data. Its adoption has outpaced its security posture — and its design makes that gap consequential.

Mature middleware such as REST or gRPC separates transport, authentication, and execution into distinct layers with clear boundaries. MCP does not. It merges reasoning and control flow within a shared semantic context: context, metadata, and executable instructions coexist without strong isolation. That fluidity is what makes MCP powerful, and it is also what blurs the trust boundary an attacker needs to cross.

The most prevalent and impactful client-side MCP vulnerability is the **Tool Poisoning Attack**, first documented by Invariant Labs in 2025. Malicious instructions are embedded in a tool's *description metadata* — text that is invisible to the user but fully visible to the model, which follows it. A poisoned tool can exfiltrate data and, worse, override the instructions of other trusted tools in the same context, leading to a complete compromise of the agent even with respect to infrastructure the operator trusts. Related techniques have followed: **rug pulls**, where a tool's definition changes after it has been approved; **shadowing**, where one server's tool description hijacks the priority or behavior of another's; and **preference manipulation**, where crafted descriptions win tool-call priority.

The exposure is not hypothetical. Through mid-2025 and into 2026, researchers disclosed that leading developer environments — including Cursor, Claude Code, Gemini CLI, GitHub Copilot, and Amazon Q — auto-execute project-defined MCP servers with developer-level operating-system privileges and no process isolation. Named CVEs followed, including an authentication gap in the MCP Inspector proxy and an OS command-injection path in a widely used MCP bridge. Systemic-flaw disclosures put hundreds of thousands of servers, and downloads numbering in the hundreds of millions, in scope.

The most useful framing to emerge comes from Microsoft's security guidance in 2026: **treat a change to a tool's description as equivalent to a dependency update** — a modification to a software artifact that directly influences agent behavior and therefore warrants review before it reaches production. The proposed controls — signed tool manifests, automated scanning of tool metadata for embedded instructions, and dynamic scoping that limits an agent to only the tools a given session needs — all point in the same direction as the rest of this paper: the tool an agent trusts, and the action it takes, must be inspected at the moment of use, not assumed safe because it was safe yesterday.

5. The security model that's emerging: identity, and then runtime

Faced with this surface, the industry has converged, through 2026, on a layered model. The most useful way to hold it is as three layers that mature organizations combine, because no single product covers all three.

Layer one is identity and access. Treat each agent as a non-human identity with a defined owner, a stated purpose, and a scoped, least-privilege set of permissions. This is where much of the enterprise investment has gone: agent identity objects, lifecycle management, and the discipline of giving an agent only the access its function requires. Microsoft's Entra Agent ID and its associated management tooling are the most visible expression of this movement. Identity governance, as one analysis put it, has become a control plane rather than a feature set. Identity answers a specific, essential question: **what may this agent reach?**

Layer two is runtime protection. This layer intercepts prompt injection, data leakage, and unauthorized actions *as they happen* — inspecting not what an agent is permitted to do, but what it is actually doing, in the moment, with the access identity granted it.

Layer three is governance and observability — policy, ownership, and the audit trail that ties behavior to a record, increasingly for regulatory reasons.

The critical insight — and the reason all three layers are necessary — is that **identity governance is table stakes but not sufficient.** Deterministic policy at the entitlement layer defines the boundary of what an agent *can* access. But agents are non-deterministic. An agent can stay entirely within its permitted access boundary and still be steered into doing something harmful. EchoLeak is the proof: the Copilot involved was operating within its legitimate permissions the entire time. Nothing about its identity or entitlements was violated. It was redirected, by content it was trusted to read, into using that legitimate access for exfiltration. No identity control would have stopped it, because no identity control inspects the *action*.

This is why the leading analysis of the space concludes that no single vendor today fully solves the runtime problem across deterministic governance, non-deterministic behavioral analysis, and non-deterministic governance. Identity tells you what an agent may reach. Only runtime inspection, on the actual traffic, tells you whether what it is doing right now — with what it is allowed to reach — is actually safe.

There is an architectural corollary worth stating plainly, because it points to where enforcement is most effective. For an indirect prompt-injection attack like EchoLeak to succeed, three conditions must hold simultaneously: the agent must **ingest untrusted external content**, it must **have access to internal or sensitive data**, and it must **be able to communicate externally**. Break any one of the three and the attack fails. Of the three, the one most cleanly enforceable in real time is the third — the external communication, the egress. An enforcement point that sits on the wire, watches the actual action, and can hold or block the exfiltration at the moment it is attempted is enforcing exactly where the attack chain is breakable.

6. The compliance clock

None of this is only an engineering concern anymore. Regulation has begun to require the very capabilities this threat model demands. The European Union's AI Act brings high-risk obligations into force in August 2026 —

including automatic recording of events over a system's lifetime, which is, in effect, an audit-trail mandate for agent behavior. The Colorado AI Act becomes enforceable in June 2026. Depending on context, the NIST AI Risk Management Framework, ISO/IEC 42001, GDPR's provisions on automated decision-making, and DORA all apply. The practical implication is that an organization deploying agents will increasingly need not only to control what they do, but to *prove* what they did — with a record that holds up.

7. What this means for anyone deploying agents

The agentic attack surface is real, it is documented, and it is expanding fastest at the newest layers — tool trust and inter-agent communication. Three conclusions follow for anyone putting agents into production.

Map before you deploy. Find every agent already running in your environment, including the ones no one registered. Know how many there are, where they run, what data and tools each can reach, and who owns each one. You cannot govern what you have not found, and industry data suggests most organizations cannot yet distinguish agent activity from human activity at all.

Combine identity with runtime; do not confuse one for the other. Scope each agent as a least-privilege non-human identity — that is the necessary first layer. But do not mistake a permission boundary for protection. Add a runtime layer that inspects the actual action, on the actual traffic, and can stop the dangerous ones as they happen. The two are complementary, and the incidents of the last eighteen months show that the second is where the visible failures occurred.

Keep the evidence. Assume you will have to prove what an agent did — to an auditor, a regulator, or yourself after an incident. Build for a tamper-evident record from the start, not after the fact.

The moment an agent can act for you is the moment it can act in ways you did not intend. The security model that matches this reality is not the one that governs what a model says. It is the one that watches what an agent does.

Crawdad Security builds runtime security for AI agents: a local, on-device gateway that inspects every action an agent takes on the wire, enforces policy you can read, keeps raw content on the machine, and writes a tamper-evident record of every decision. It operates at layer two of the model described above — complementary to agent identity, and focused on the action itself. Learn more at getcrawdad.dev.

This paper cites public research and disclosures, including the OWASP Top 10 for Agentic Applications 2026, and publicly documented incidents (EchoLeak / CVE-2025-32711; CamoLeak / CVE-2025-59145; ForcedLeak; the July 2025 Replit incident). It is intended as an industry orientation, not vendor benchmarking.