AINARY RESEARCH REPORT NO. AR-006

The AI Agent Security Playbook

What Attackers Already Know That Defenders Don't

Florian Ziesche — Ainary Ventures

February 2026

Overall Confidence: 78%

CONTENTS

01 Executive Summary

02 Methodology

03 The Attack Surface Nobody Modeled

04 Prompt Injection — The Unsolvable Problem

05 Memory Poisoning — The Long Game

06 Tool Use Exploitation — When Agents Have Keys

07 Multi-Agent Contagion — One Compromise, Total Infection

08 Supply Chain Attacks on Agent Frameworks

09 What Actually Works

10 Beipackzettel

11 Claim Register

12 References

1. Executive Summary

- Every published prompt injection defense (12/12) has been broken by adaptive attacks from researchers at Meta, OpenAI, Anthropic, and DeepMind[1]
- Memory injection attacks achieve >95% success rates against production agent memory systems, creating persistent backdoors that survive session resets[2]
- Multi-agent system hijacking succeeds 45–64% of the time across AutoGen, CrewAI, and MetaGPT — with zero inter-agent trust verification[3]
- No production memory framework implements provenance tracking, integrity checks, or confidence scoring per memory entry[10]
- Agent security requires architectural constraints (privilege separation, deterministic guardrails, kill switches) — not better prompt engineering

Keywords: AI Agent Security, Prompt Injection, Memory Poisoning, Tool Exploitation, Multi-Agent Attacks, Red Teaming, NIST AI RMF

2. Methodology

This report synthesizes primary research from peer-reviewed papers (arXiv), CVE databases (NVD), and publications from NIST, OWASP, and MITRE. Industry data comes from practitioner surveys (Okta, Vectra AI, World Economic Forum) and framework documentation analysis.

The research pipeline followed a structured process: 15 research briefs covering adversarial AI, agent memory, agent protocols, failure taxonomies, and regulatory frameworks were produced independently. A synthesis phase cross-referenced findings across briefs, identifying contradictions and compound effects. Gap research targeted memory governance, credential management, and multi-agent contagion.

Confidence ratings follow a three-tier system: **High** (3+ independent sources, peer-reviewed or primary data), **Medium** (1–2 sources, plausible but not independently confirmed), **Low** (single secondary source, methodology unclear). Every quantitative claim carries its source and confidence inline.

Limitations: The agent security field is moving fast. Several findings rely on pre-print papers not yet peer-reviewed through traditional journal processes (though arXiv pre-prints from top AI labs carry significant weight). Production incident data is scarce because most agent deployments are recent and organizations do not publicly disclose agent-specific security failures.

3. The Attack Surface Nobody Modeled (Confidence: High)

A chatbot has one attack surface. An agent has seven — and they compound.

The thesis of this report is simple: every defense deployed today was designed for chatbots, not agents. Agents have tools, memory, and network access — the attack surface is 10x larger. The evidence supports this claim across every dimension I examined.

Evidence

A chatbot receives a prompt and returns text. That is one attack surface: the prompt. An agent receives a prompt, retrieves data from external sources, consults persistent memory, calls APIs, executes code, communicates with other agents, and manages credentials. Each of these is a distinct attack vector.

Exhibit 1: Agent vs. Chatbot Attack Surface Comparison

ATTACK SURFACE	CHATBOT	AGENT
Direct prompt input	✅	✅
Indirect prompt (retrieved data)	❌	✅
Persistent memory	❌	✅
Tool/API calls	❌	✅
Inter-agent communication	❌	✅
Credential/key access	❌	✅
External data sources (RAG)	Limited	✅

Source: Author analysis based on OWASP Top 10 for LLM Applications (2025) [4], MITRE ATLAS [6]

The OWASP Top 10 for LLM Applications (2025 edition) lists prompt injection as the #1 risk.[4] But the taxonomy was designed for single-model applications. Agent-specific attack vectors — memory poisoning, tool chain exploitation, inter-agent contagion — are absent or underspecified.

MITRE ATLAS added "LLM Prompt Injection" (AML.T0051) and related techniques in 2024, but does not model multi-agent propagation or persistent memory corruption as distinct attack paths.[6]

Interpretation

No widely adopted threat model exists for autonomous AI agents. Security teams are using chatbot threat models for agent deployments. This is like using a firewall ruleset designed for a static website to protect a Kubernetes cluster. The seven attack surfaces listed in Exhibit 1 do not just add up — they multiply. A prompt injection that is harmless in a chatbot becomes catastrophic when it can trigger tool calls, corrupt memory, and propagate to other agents.

The compound effect matters most. Consider a realistic attack chain: an attacker plants a poisoned document in a public data source (attack surface: external data). The agent retrieves it during RAG (attack surface: indirect prompt). The poisoned instruction tells the agent to modify its memory (attack surface: persistent memory). The corrupted memory causes the agent to misuse a tool on the next session (attack surface: tool/API calls). The tool call leaks credentials (attack surface: credential access). The leaked credentials are used to compromise a connected agent (attack surface: inter-agent communication). Six surfaces, one attack chain, and each transition is individually documented in the research literature.

I want to be precise about what is evidence and what is extrapolation here. Each individual attack vector (memory injection, tool exploitation, inter-agent hijacking) has peer-reviewed documentation. The full six-step chain is my synthesis — I have not found a documented case of a complete compound attack in production. But the absence of documented cases likely reflects the immaturity of agent deployments and the lack of agent-specific incident reporting, not the absence of risk.

> **So What?** *Every security team deploying agents needs a new threat model. The chatbot model is not wrong — it is incomplete. Start with Exhibit 1 and map which surfaces your agent exposes. Then model the compound chains between surfaces — that is where the real risk lives.*

> **What would invalidate this?** *If a production agent framework emerged that architecturally isolated each attack surface (e.g., hardware-level separation between prompt processing and tool execution), the compound risk would be significantly reduced. No such framework exists today.*

4. Prompt Injection — The Unsolvable Problem (Confidence: High)

Prompt injection is not a bug to be patched. It is an inherent property of systems that mix instructions and data in the same channel.

Evidence

Direct prompt injection — where a user manipulates the system prompt through their input — remains trivially exploitable. The "Defeating Prompt Injections by Design" paper (arXiv:2510.09023) is the most comprehensive study to date: a 14-author team from Meta, OpenAI, Anthropic, and Google DeepMind tested 12 defense categories:[1]

- Perplexity-based detection
- Input/output classifiers
- Paraphrasing defenses
- Instruction hierarchy
- Sandwich defenses
- XML/delimiter tagging
- Known-answer detection
- Spotlighting
- Prompt shields (Azure)
- LLM-as-judge
- Fine-tuned safety models
- Constitutional AI guardrails

Result: **12/12 broken.** Not theoretically — with working code. Every defense was defeated with at most two adaptive attack iterations.[1]

Indirect prompt injection is worse. The agent retrieves external data — web pages, emails, documents, API responses — that contains adversarial instructions. The agent cannot distinguish between "data to process" and "instructions to follow" because both arrive as text in the same context window.

Documented real-world cases:

- **CVE-2025-32711 (EchoLeak):** Hidden instructions in documents retrieved by Microsoft Copilot caused zero-click data exfiltration — the agent silently sent user data to attacker-controlled

endpoints. No user interaction required.[5]

- **Grok RAG Poisoning (2025):** Manipulated source documents caused Grok to generate false outputs that spread on X/Twitter.[7]
- **ChatGPT Plugin Exploits (2024):** Researchers demonstrated that a malicious plugin could inject instructions into ChatGPT's context, hijacking subsequent tool calls.[8]

73% of AI deployments have prompt injection vulnerabilities according to OWASP.[4] This number carries medium confidence — the methodology behind the specific percentage is not fully transparent — but the directional finding is consistent with every other data source I reviewed.

Interpretation

The fundamental issue is architectural: LLMs process instructions and data in the same modality (natural language tokens). There is no hardware-level separation equivalent to kernel/user space in operating systems. Every proposed "fix" is a heuristic that can be circumvented because the model cannot fundamentally distinguish instruction from data.[1]

This does not mean defenses are useless. It means defenses must be layered and must assume breach — exactly the zero-trust paradigm that network security adopted 15 years ago. The shift required is from "prevent prompt injection" to "limit the blast radius when prompt injection succeeds."

> **So What?** *Stop investing in better prompt-level defenses as your primary security strategy. Instead, invest in constraining what a compromised agent can do. Privilege separation, scoped tool access, and deterministic guardrails at the infrastructure layer — not the prompt layer — are where security budget should go. The mental model shift: from "prevent injection" to "survive injection."*

To be concrete: if your agent can send emails, the security question is not "can I prevent prompt injection?" (you cannot, reliably). The question is "when prompt injection succeeds, can the agent send emails to arbitrary recipients?" If yes, you have an architectural problem that no prompt defense will fix.

> **What would invalidate this?** *A fundamental architectural breakthrough that separates instructions from data at the model level (not heuristic-based). Research into this exists — including proposals for dual-LLM architectures where one model processes data and another*

processes instructions — but no viable approach has been demonstrated at production scale. If such a breakthrough occurred, it would change the economics of the entire field.

📖 5. Memory Poisoning — The Long Game (Confidence: High)

Memory poisoning is the agent equivalent of a persistent backdoor. Once planted, it survives context window resets and influences every future interaction.

Evidence

Unlike prompt injection (which is ephemeral), memory poisoning creates persistent compromise. The attacker modifies the agent's long-term memory — stored in vector databases, knowledge graphs, or structured memory systems — so that the poisoned information is retrieved and trusted in all future sessions.

The MINJA attack (arXiv:2503.03704) demonstrated **>95% injection success** against RAG-based agent memory systems.[2] The attack crafts documents that, when ingested into the memory store, contain adversarial instructions that override the agent's intended behavior whenever the poisoned memory is retrieved. Key finding: the attack persists across sessions. Unlike prompt injection, which requires the attacker to be present in the conversation, memory poisoning is "fire and forget."

MemoryGraft (arXiv:2512.16962) goes further: researchers demonstrated planting persistent false experiences in agent memory that the agent treats as its own past interactions.[9] The agent cannot distinguish between genuine memories and implanted ones — there is no memory integrity verification in any production memory framework.

Exhibit 2: Memory Framework Security Features

FRAMEWORK	PROVENANCE TRACKING	INTEGRITY CHECKS	CONFIDENCE PER MEMORY	SELECTIVE FORGETTING	AUDIT TRAIL
Letta (MemGPT)	❌	❌	❌	Partial	❌
Mem0	❌	❌	❌	❌	Partial
Zep	❌	❌	❌	❌	Partial
LangMem	❌	❌	❌	❌	❌
A-Mem	❌	❌	❌	❌	❌

Source: Framework documentation review, February 2026 [10]

No production memory framework implements memory provenance, integrity verification, or confidence scoring per memory entry. Every framework trusts all stored memories equally. This is the equivalent of running a database without access controls.

The Adversarial Memory Spiral

When memory poisoning combines with other vulnerabilities, it creates a self-reinforcing attack loop:

1. MINJA injects poisoned memory (>95% success)[2]
2. Agent's verbalized confidence reports high certainty (VCE is systematically biased — arXiv:2602.00279)[11]
3. Poisoned output passes to other agents (no inter-agent trust verification)
4. HITL alert fires but is ignored (67% alert ignore rate — Vectra 2023, n=2,000 analysts)[12]
5. Agent reinforces poisoned memory based on "successful" interaction
6. Loop repeats with increasing conviction

Each step is independently documented with high-confidence sources. The full chain has not been observed in the wild — but every link is empirically validated. This distinction matters: I am describing a compound risk, not a documented incident.

Interpretation

Memory is the agent's most dangerous capability from a security perspective. It turns ephemeral attacks into persistent ones. The complete absence of memory security primitives across all five major frameworks (Exhibit 2) suggests the problem is not being addressed at the ecosystem level.

> **So What?** *If you deploy agents with persistent memory, treat the memory store as a security-critical system. Implement write-ahead logging, provenance tracking, and integrity checks at the storage layer. Do not wait for framework vendors to solve this.*
>
> **What would invalidate this?** *If memory frameworks shipped with built-in provenance tracking and cryptographic integrity verification, the attack surface would shrink significantly. This is technically feasible — it just is not being built.*
>
> **Positive counterexample:** *Letta (MemGPT) offers partial selective forgetting, indicating that at least one framework team is thinking about memory governance. This is early but directionally*

> *correct. The gap between "partial selective forgetting" and "full memory integrity with provenance" is large, but the fact that any framework considers memory management as a first-class concern is encouraging. I expect memory security to become a competitive differentiator for agent platforms within 12 months.*

The memory poisoning risk also scales with agent longevity. An agent that runs for a single task and is discarded has minimal memory risk. An agent that runs continuously for weeks or months, accumulating memories from hundreds of interactions, has a proportionally larger memory attack surface. The trend in the industry is toward longer-lived, more autonomous agents — which means this problem gets worse, not better.

---

6. Tool Use Exploitation — When Agents Have Keys (Confidence: High)

When an agent can call APIs, execute code, and manage files, prompt injection escalates from "wrong answer" to "unauthorized action."

Evidence

The Model Context Protocol (MCP) standardizes how agents connect to external tools. MCPTox (arXiv:2508.14925) demonstrated that attackers can embed malicious instructions in MCP tool descriptions.[13] When an agent reads the tool's metadata to decide how to use it, the poisoned description hijacks the agent's behavior. This is supply chain poisoning at the tool level.

Tool-calling fails 3–15% of the time in production (practitioner reports — single source, low confidence on the specific range).[14] But security-relevant failures are different from reliability failures:

- **Credential leakage:** 23% of IT professionals report agent credential leaks (Okta survey)[15]
- **Excessive permissions:** Agents routinely receive broader API scopes than needed because fine-grained agent-specific IAM policies do not exist in most cloud providers
- **Unvalidated outputs:** Agents pass tool outputs directly into their context without sanitization — a classic injection vector
- **Confused deputy attacks:** An agent authorized to call Tool A is tricked (via prompt injection) into calling Tool B with Tool A's credentials

Only **10% of organizations have a non-human identity strategy** (World Economic Forum).[16] Agents are accessing production APIs with shared service accounts, hardcoded tokens, or overly broad OAuth scopes. When an agent is compromised, the blast radius is determined by its credential scope — which is usually "everything."

Exhibit 3: Agent Credential Anti-Patterns

ANTI-PATTERN	PREVALENCE	RISK
Shared service accounts	High	Lateral movement after compromise
Hardcoded API keys in agent config	High	Key extraction via prompt injection

ANTI-PATTERN	PREVALENCE	RISK
Overly broad OAuth scopes	Very High	Privilege escalation
No credential rotation	High	Persistent access after compromise
No per-action authorization	Very High	Confused deputy attacks

Source: Okta [15], WEF [16], OWASP LLM Top 10 [4]

Interpretation

The tool layer is where prompt injection becomes dangerous. A chatbot that gives a wrong answer is annoying. An agent that executes an unauthorized API call, leaks credentials, or modifies production data is a security incident. The combination of excessive permissions and missing per-action authorization means that a single successful prompt injection can cascade into full system compromise.

> **So What?** *Implement least-privilege for every agent tool connection. Each tool call should require scoped, short-lived credentials. Validate all tool outputs before they enter the agent's context. Treat MCP tool servers like untrusted third-party code — because that is what they are.*

> **What would invalidate this?** *If cloud providers shipped agent-specific IAM primitives (per-action authorization, automatic scope reduction, credential rotation tied to agent sessions), the blast radius of tool exploitation would shrink dramatically. AWS, Google Cloud, and Azure all have sophisticated IAM systems — but none currently offer agent-specific identity primitives. The first cloud provider to ship "agent IAM" will have a meaningful security advantage.*

> **Positive counterexample:** *Some teams are implementing custom authorization layers on top of MCP that validate each tool call against an allowlist before execution. This is manual and fragile, but demonstrates that the problem is solvable at the infrastructure level. The pattern: do not trust the agent to self-regulate tool use. Enforce constraints externally.*

7. Multi-Agent Contagion — One Compromise, Total Infection (Confidence: High)

Multi-agent systems have no immune system. Compromising one agent propagates to all connected agents because inter-agent messages are trusted by default.

Evidence

The multi-agent system (MAS) hijacking paper (arXiv:2503.12188) tested attacks against three major multi-agent frameworks:[3]

Exhibit 4: MAS Hijacking Success Rates

FRAMEWORK	HIJACKING SUCCESS RATE	ATTACK TYPE
AutoGen	45%	Task injection via agent message
CrewAI	55%	Role confusion + instruction override
MetaGPT	64%	Workflow manipulation

Source: arXiv:2503.12188 [3]

The attack pattern: compromise one agent in a multi-agent pipeline, then use that agent's trusted position to inject instructions into downstream agents. Because inter-agent communication is treated as trusted input (it comes from "inside the system"), standard prompt injection defenses do not apply.

The A2A protocol (Google → Linux Foundation) authenticates **systems** via OAuth/OpenID but does not verify **message provenance or intention**.[17] Knowing that a message came from an authenticated agent says nothing about whether that agent has been compromised. Authentication is not integrity verification.

In multi-agent systems processing concurrent tasks, a single compromised agent can influence every other agent within one task cycle. There is no quarantine mechanism, no anomaly detection on inter-agent message content, and no rollback capability for poisoned decisions.

Interpretation

This mirrors the systemic risk pattern from the 2008 financial crisis: Agent A trusts Agent B trusts Agent C. When Agent C is compromised, the trust chain propagates the compromise backwards. There is no "circuit breaker."

The 45–64% hijacking success rates are from academic settings. Production multi-agent deployments may have additional defenses or different configurations. But the fundamental architectural vulnerability — implicit trust of inter-agent messages — is present in every framework I examined.

> **So What?** *If you run multi-agent systems, implement message validation outside the LLM context. Use a deterministic validation layer that checks message content against expected patterns. Build circuit breakers that can isolate a compromised agent.*

> **What would invalidate this?** *If multi-agent frameworks implemented cryptographic message signing with content integrity verification (not just sender authentication), contagion risk would be significantly reduced. The A2A protocol could be extended to include this — but currently does not.*

The deeper issue is conceptual: the multi-agent community has borrowed the microservices architecture pattern (small, independent services communicating via APIs) without borrowing the security patterns that make microservices safe (service mesh, mutual TLS, request-level authorization, circuit breakers). The architectural primitives exist in the infrastructure world. They have not been adapted for agent-to-agent communication.

I want to flag a nuance: the 45–64% hijacking rates[3] were measured in default configurations. Teams that implement additional guardrails (output validation, restricted communication channels, human approval for sensitive actions) likely see lower rates. But the default configuration is what most deployments use — especially in the early adoption phase where speed-to-market dominates security considerations.

---

8. Supply Chain Attacks on Agent Frameworks (Confidence: Medium)

The agent framework ecosystem has the same supply chain vulnerabilities as npm/PyPI — but compromised packages can autonomously execute actions.

Evidence

Agent frameworks are built on deep dependency trees. Known vulnerabilities in LangChain (as of 2023 — no published 2025/2026 agent framework CVEs identified, which likely indicates under-reporting rather than absence of vulnerabilities):

- **CVE-2023-36258:** Arbitrary code execution via the `PALChain` module[18]
- **CVE-2023-36281:** SQL injection via the `SQLDatabaseChain`[19]
- **CVE-2023-39659:** Server-Side Request Forgery (SSRF) via multiple components[20]
- **AutoGen** executes LLM-generated code by default — any prompt injection becomes code execution[21]

MCP tool servers are the agent equivalent of npm packages. Anyone can publish an MCP server. There is no code review, no signing, no sandbox. When an agent connects to a malicious MCP server:

1. The tool description can contain prompt injection (MCPTox[13])
2. The tool can return poisoned data
3. The tool can log all agent requests (including credentials and user data)
4. The tool can modify its behavior after initial trust is established

Standard supply chain attacks — typosquatting, dependency confusion, compromised maintainer accounts — apply directly. The difference: a compromised agent framework dependency does not just steal data. It can **act** — send emails, modify files, call APIs, communicate with other agents.

Interpretation

The supply chain section carries medium confidence because it relies partly on 2023 CVEs for LangChain with no published 2025/2026 agent framework CVEs found. This does not mean vulnerabilities do not exist — it likely indicates under-reporting. The agent framework ecosystem is young, and security auditing has not caught up.

So What? *Audit your agent framework dependencies with the same rigor you apply to production web applications. Pin versions, verify checksums, and sandbox MCP tool servers. Treat every external tool connection as a potential attack vector.*

What would invalidate this? *If MCP adopted a signing and review process similar to mobile app stores, and if agent frameworks sandboxed all external tool execution by default, supply chain risk would drop significantly.*

Positive counterexample: *The move of A2A from Google to the Linux Foundation[17] signals that the industry recognizes the need for governance over agent communication standards. Foundation stewardship typically brings security review processes that single-vendor projects lack. Similarly, the Anthropic MCP specification, while lacking security primitives today, provides a standardization surface on which security features could be built. Standardization is a prerequisite for security — you cannot secure what you have not defined.*

---

9. What Actually Works (Confidence: Medium)

Based on the evidence in this report, effective agent security requires architectural constraints, not pattern matching. The zero-trust model must extend to agent internals.

> **Scope:** This framework applies primarily to autonomous agents with tool access, persistent memory, and multi-agent coordination. Agents that perform single tasks with no persistent state have a narrower attack surface and may not require all controls described here.

Security Theater

Exhibit 5: Defense Effectiveness Assessment

DEFENSE	STATUS	WHY
System prompt hardening	❌ Theater	Broken by adaptive attacks (12/12) [1]
Input/output keyword filtering	❌ Theater	Trivially bypassed via encoding, synonyms, multilingual attacks
Single-layer guardrail models	❌ Theater	Same vulnerability class as the model they protect
"AI safety" fine-tuning alone	❌ Theater	Training-time alignment insufficient against inference-time attacks [1]
Perplexity-based detection	❌ Theater	High false positive rate, adversarially evadable [1]

Architectural Constraints That Work

- **Privilege separation:** Agents should operate with minimum necessary permissions. Each tool call should require explicit, scoped authorization — not blanket API access.

- **Memory integrity verification:** Cryptographic signing of memory entries with provenance tracking. Treat memory like a write-ahead log, not a trust store.
- **Inter-agent message validation:** Messages between agents should be verified outside the LLM context — using a separate, deterministic validation layer.
- **Deterministic guardrails:** Hard-coded rules that cannot be overridden by the LLM (e.g., "never execute `rm -rf`" enforced at the tool layer, not the prompt layer).

Detection and Response

- **Behavioral baselines:** Monitor agent actions (tool calls, API patterns, memory writes) against expected behavior profiles. Anomaly detection on actions, not on text.
- **Kill switches:** Infrastructure-level ability to halt agent execution immediately. Not a prompt that says "stop" — a process kill.
- **Audit trails:** Every agent action logged immutably with full context. Not for compliance — for incident response.

Red Teaming Frameworks

- **NIST AI RMF (AI 100-1):** Four pillars — Govern, Map, Measure, Manage. The January 2026 RFI on agent-specific security signals that NIST recognizes existing frameworks are insufficient for agents.[22]
- **MITRE ATLAS:** Adversarial threat landscape for AI systems. Includes LLM-specific techniques but needs extension for multi-agent scenarios.[6]
- **OWASP LLM Top 10 (2025):** Best current taxonomy for single-model risks. Does not cover agent-specific vectors adequately.[4]

Agent-specific red teaming must test the full chain: prompt injection → tool exploitation → memory corruption → inter-agent propagation. Single-vector testing misses compound attacks.

The Regulatory Gap

Exhibit 6: Regulatory Framework Comparison for Agent Security

FRAMEWORK	AGENT-SPECIFIC PROVISIONS	STATUS
EU AI Act	None — applies general high-risk provisions	Enforcement Aug 2026

FRAMEWORK	AGENT-SPECIFIC PROVISIONS	STATUS
NIST AI RMF 1.0	None — single-model focus	RFI for agent extensions Jan 2026
NIST CAISI	Building agent security standards	RFI deadline Mar 2026
OWASP LLM Top 10	Partial — prompt injection, insecure plugins	2025 edition
MITRE ATLAS	Partial — LLM techniques, no multi-agent	Ongoing updates
ISO 42001	General AI management, not agent-specific	Published 2023

Sources: EU AI Act [23], NIST [22], OWASP [4], MITRE [6], ISO [24]

The EU AI Act (enforcement for high-risk systems from August 2026) requires human oversight (Article 14). But empirical evidence shows HITL oversight fails at scale: **67% of security alerts are ignored** (Vectra 2023, n=2,000 analysts).[12] The regulation mandates a control that research proves does not work at production scale. NIST is more pragmatic: the January 2026 RFI explicitly asks for input on agent-specific security requirements, acknowledging that existing frameworks were designed for single-model systems.[22]

> **So What?** *Based on this evidence, build your agent security stack around architectural constraints, not prompt-level defenses. Use the NIST AI RMF as your governance backbone but extend it with agent-specific controls. Do not wait for regulations — they are 12–18 months behind the technology.*

> **What would invalidate this?** *If prompt-level defenses achieved a breakthrough that made them reliably unbreakable (contradicting the 12/12 finding), the architectural approach would become less critical. Nothing on the horizon suggests this.*

A Practical Security Stack

For teams deploying agents today, here is the minimum viable security stack I would recommend based on this research:

1. **Privilege separation at the tool layer.** Every tool call scoped to minimum necessary permissions. Enforced outside the LLM, in deterministic code.
2. **Memory integrity.** Cryptographic hashing of memory entries at write time. Provenance metadata (source, timestamp, confidence) on every memory record.
3. **Behavioral monitoring.** Log every agent action (not just prompts). Build baselines. Alert on deviations in tool call patterns, memory write frequency, and inter-agent message volume.
4. **Kill switches.** Infrastructure-level (process kill, network isolation), not prompt-level ("please stop"). Tested regularly.
5. **Red teaming the full chain.** Test prompt injection → tool exploitation → memory corruption → inter-agent propagation as a single attack scenario. Single-vector testing gives false confidence.

This is not a complete security framework. It is the minimum set of controls that addresses the specific vulnerabilities documented in this report. Each control maps directly to an evidence-backed attack vector.

Beipackzettel

- **Overall Confidence:** 78%
- **Sources:** 13 primary (peer-reviewed papers, CVE databases, framework documentation), 11 secondary (industry reports, practitioner analysis)
- **Strongest Evidence:** 12/12 prompt injection defenses broken (arXiv:2510.09023 — 14 authors from Meta, OpenAI, Anthropic, Google DeepMind) [1]; MINJA >95% memory injection success (arXiv:2503.03704) [2]
- **Weakest Point:** Supply chain attack section relies on 2023 CVEs for LangChain; no published 2025/2026 agent framework CVEs found (does not mean they do not exist — likely indicates under-reporting)
- **What would invalidate this report?** A fundamental architectural breakthrough that separates instructions from data at the model level (not heuristic-based). No such breakthrough is on the horizon.
- **Methodology:** Multi-agent research pipeline. Primary research from arXiv, CVE databases, NIST/OWASP/MITRE publications. Cross-referenced against Ainary Research Briefs #10 (Adversarial Agents), #12 (Agent Memory), #13 (Agent Protocols). 15 independent research briefs synthesized, with gap research on memory governance, credential management, and multi-agent contagion.
- **This report was created with a Multi-Agent Research System.**

Claim Register

Exhibit 7: Claim Register

#	CLAIM	VALUE	SOURCE	CONFIDENCE
1	All published prompt injection defenses broken	12/12	arXiv:2510.09023 [1]	**High**
2	Memory injection success rate (MINJA)	>95%	arXiv:2503.03704 [2]	**High**
3	MAS hijacking success rate	45–64%	arXiv:2503.12188 [3]	**High**
4	AI deployments with prompt injection vulnerabilities	73%	OWASP [4]	**Medium**
5	Zero-click exfiltration via Copilot	CVE-2025-32711	Microsoft/NVD [5]	**High**
6	SOC alerts ignored	67%	Vectra 2023 (n=2,000) [12]	**High**
7	Agent credential leaks reported	23%	Okta [15]	**Medium**
8	Organizations with non-human identity strategy	10%	WEF [16]	**Medium**
9	No memory framework has integrity checks	0/5 frameworks	Framework documentation [10]	**High**
10	LangChain arbitrary code execution	CVE-2023-36258	NVD [18]	**High**
11	VCE systematically biased	Confirmed	arXiv:2602.00279 [11]	**High**

#	CLAIM	VALUE	SOURCE	CONFIDENCE
12	MCPTox tool poisoning via descriptions	Demonstrated	arXiv:2508.14925 [13]	**High**

References

[1] Debenedetti, E., et al. (2025). "Defeating Prompt Injections by Design." arXiv:2510.09023. Meta, OpenAI, Anthropic, Google DeepMind. 14 authors.

[2] Chen, Z., et al. (2025). "MINJA: Memory INJection Attacks against Retrieval-Augmented Generation." arXiv:2503.03704.

[3] Gu, Y., et al. (2025). "Hijacking Multi-Agent Systems." arXiv:2503.12188.

[4] OWASP. (2025). "OWASP Top 10 for Large Language Model Applications." Version 2.0.

[5] Microsoft / NVD. (2025). "CVE-2025-32711: EchoLeak — Information Disclosure in Microsoft Copilot."

[6] MITRE. (2025). "ATLAS — Adversarial Threat Landscape for AI Systems." AML.T0051.

[7] Documented in Ainary Research Brief #5 (Agent Failures). Original reporting: multiple sources, 2025.

[8] Greshake, K., et al. (2023). "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection." arXiv:2302.12173.

[9] Zhang, Y., et al. (2025). "MemoryGraft: Persistent False Memories in AI Agents." arXiv:2512.16962.

[10] Framework analysis: Letta, Mem0, Zep, LangMem, A-Mem documentation. Reviewed February 2026.

[11] Tao, Z., et al. (2026). "Verbalized Confidence Estimation is Biased." arXiv:2602.00279.

[12] Vectra AI. (2023). "State of Threat Detection." n=2,000 security analysts.

[13] Wang, X., et al. (2025). "MCPTox: Tool Poisoning Attacks on Model Context Protocol." arXiv:2508.14925.

[14] Hannecke, M. (2025). "Tool-calling failure rates in production." Practitioner blog (single source).

[15] Okta. (2025). "State of Machine Identity Security." IT professional survey.

[16] World Economic Forum. (2025). "Non-Human Identity Management."

[17] Google. (2025). "Agent-to-Agent (A2A) Protocol." Linux Foundation.

[18] NVD. "CVE-2023-36258: LangChain PALChain Arbitrary Code Execution."

[19] NVD. "CVE-2023-36281: LangChain SQLDatabaseChain SQL Injection."

[20] NVD. "CVE-2023-39659: LangChain Server-Side Request Forgery."

[21] Microsoft. "AutoGen Framework Documentation." Code execution enabled by default.

[22] NIST CAISI. (2026). "Request for Information: AI Agent Security." Deadline March 2026.

[23] European Parliament. (2024). "Regulation (EU) 2024/1689 — AI Act."

[24] ISO. (2023). "ISO/IEC 42001: Artificial Intelligence Management System."

Cite as: Ziesche, F. (2026). The AI Agent Security Playbook — What Attackers Already Know That Defenders Don't. Ainary Research Report, AR-006.

About the Author: Florian Ziesche is the founder of Ainary Ventures, a research and advisory firm focused on AI agent trust infrastructure. He writes about the gap between AI capability and AI trustworthiness.
