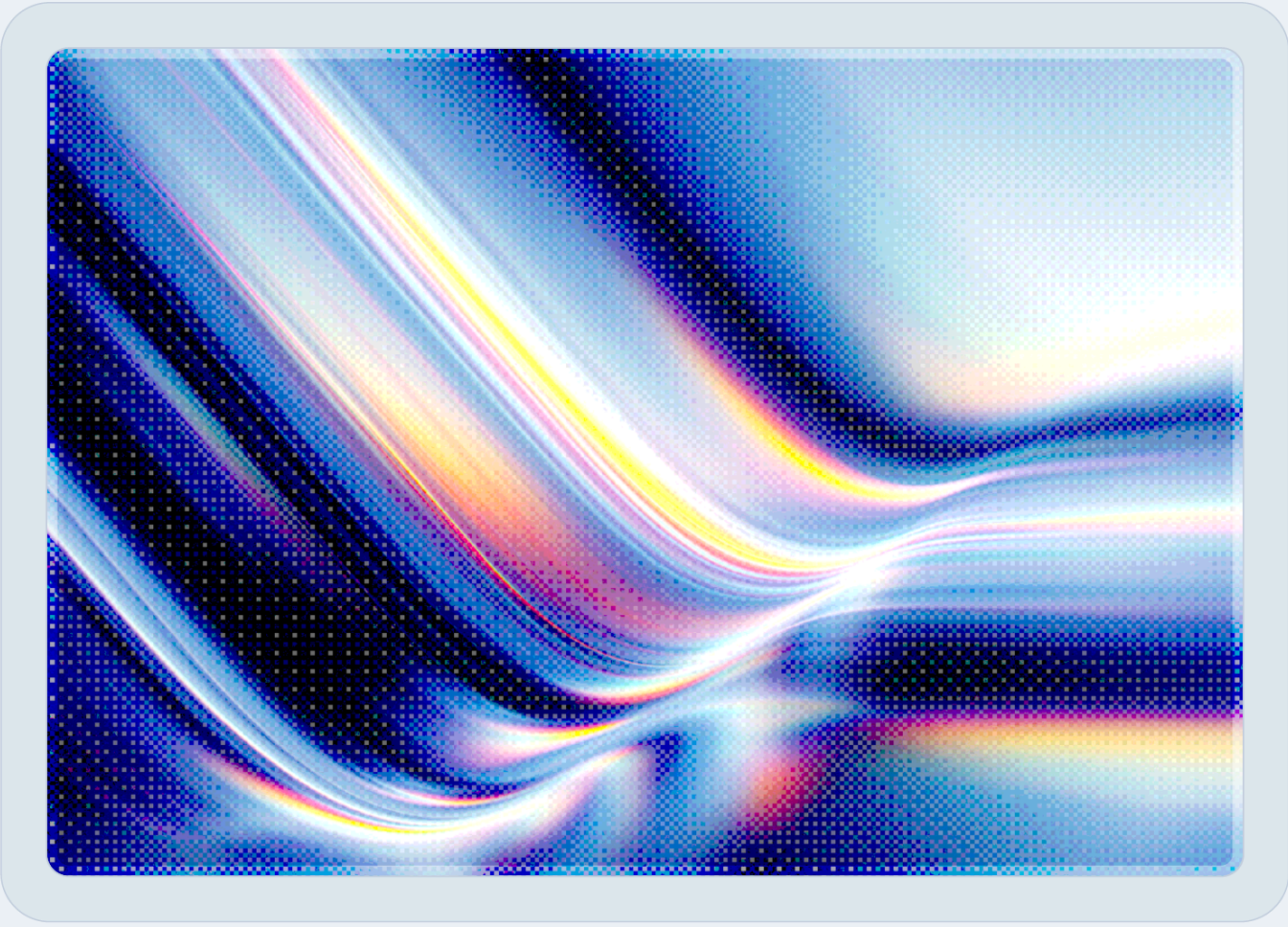
CSAI Foundation | Cloud Security Alliance

AI Coding Agents as Attack Surface: MCP, Poisoning, and Miasma

How MCP Auto-Execution, Repository Poisoning, and the Miasma Worm Turn Developer Tools into Threat Vectors

2026-06-28

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Table of Contents

Executive Summary

The rapid adoption of AI coding agents has introduced a category of security risk the industry has not previously encountered: trusted developer tooling that executes instructions autonomously, with broad access to credentials, source code, and system resources, and that reads environmental context files as first-class inputs. Adversaries have taken notice.

Three converging threat vectors now define this space. The Model Context Protocol (MCP), the primary open standard connecting AI agents to external tools, contains an architectural flaw that enables arbitrary remote code execution across all officially supported language SDKs — a vulnerability that OX Security researchers found affecting up to 200,000 vulnerable installations across a software supply chain of more than 150 million downloads [1]. Security researchers have simultaneously demonstrated that AI coding agents can be weaponized through repository configuration files — `.cursorrules`, `CLAUDE.md`, `.github/copilot-instructions.md` — using hidden Unicode characters and natural-language payload injection techniques that evade human code review and achieve command execution success rates of 84–85% against major platforms in systematic testing [2][3][7]. Most concretely, the Miasma supply chain worm, first detected in May 2026 and attributable to the TeamPCP threat group, demonstrated all of these attack primitives in a live campaign that disabled 73 Microsoft GitHub repositories and simultaneously deployed 37 malicious packages to the Python Package Index (PyPI) [4][5].

Together, these incidents establish AI coding agents as a first-class attack surface — one that requires immediate defensive action and a need to extend software supply chain security controls to govern the AI agents that developers use to build that software. This whitepaper provides security practitioners and organizational leaders with the technical context, threat actor perspective, and prioritized control recommendations needed to respond.

Introduction and Background

The Agentic Developer Tool

The shift from AI coding assistants that suggest individual lines to autonomous coding agents that plan, execute, and verify multi-step development tasks happened faster than the security industry's tooling and control frameworks could accommodate. Products such as Claude Code, GitHub Copilot (Agent mode), Cursor, Windsurf, Gemini CLI, and OpenAI Codex CLI are no longer passive autocomplete engines — they

are autonomous systems with the ability to read files, execute shell commands, call external APIs, browse the web, and commit code to version control, all with configurable — and frequently limited — human review of individual steps, depending on deployment configuration.

This capability transformation is precisely what makes these tools valuable to developers and dangerous in adversarial contexts. An agent with write access to a repository, execute permissions in a terminal, and read access to environment variables is, from a threat modeling perspective, a privileged insider — one whose instructions can potentially be injected by an attacker who gains influence over any element of the agent's context window.

What AI Agents Read

To understand the attack surface, security practitioners must understand what AI coding agents consume before they act. Every major coding agent reads and executes context from a predictable set of sources: the current repository's content and git history, project-specific instruction files (`.cursorrules` in Cursor, `CLAUDE.md` in Claude Code, `.github/copilot-instructions.md` in GitHub Copilot, `.gemini/config.md` in Gemini CLI), MCP server tool definitions and descriptions, agent skill libraries (SKILL.md files in systems such as OpenClaw), README files and code comments, and any content returned by tools the agent is permitted to invoke.

The crucial security implication is that none of these inputs are signed, hash-verified, or sandboxed from the agent's reasoning process by default. An attacker who can influence any of these sources gains a channel through which they can inject instructions into an AI system that may have shell execution permissions, credential access, and the ability to push code upstream. This is the structural precondition from which all three threat vectors examined in this paper emerge.

Threat Landscape Context

Security researchers began documenting prompt injection risks in large language models as early as 2023, but the threat environment changed qualitatively in 2025 and 2026 as agents gained tool use capabilities and deployment scale. MCP adoption on GitHub grew rapidly through 2025, with community registries indexing thousands of new server implementations [6], creating a new software ecosystem with essentially no vetting infrastructure. Agent skill marketplaces such as ClawHub emerged alongside them, offering natural-language packages for AI agents that are installable with the equivalent of a one-line configuration change.

Security vetting infrastructure for this ecosystem remains nascent: no code signing standard exists for MCP servers, registry malware scanning is absent or inconsistent, and publisher verification is minimal. Research published in January 2026 found that all major AI coding agents — Claude Code, GitHub Copilot, Cursor,

Windsurf, and others — succumb to prompt injection attacks at rates exceeding 85% when adaptive adversarial strategies are applied [20]. The OWASP Agentic Skills Top 10 has formalized Malicious Skills (AST01) as the leading threat category [8]. This situation mirrors the early days of package managers — rapid adoption of a powerful ecosystem with trust and security infrastructure still being built — with the additional dimension that the exploited components are actively reasoning about their environment and taking autonomous action.

MCP Auto-Execution: Architecture Risk Becomes Exploit

How MCP Was Designed

Anthropic introduced the Model Context Protocol in late 2024 as an open standard for connecting AI agents to external data sources, tools, and services. MCP follows a client-server architecture in which an AI agent (the MCP client) discovers and invokes tools provided by MCP servers. Tool definitions include natural-language descriptions that the agent reads when deciding which tool to invoke and how to invoke it. The protocol's design goal was interoperability: any compliant MCP server should be usable with any compliant MCP client.

MCP supports two primary transport mechanisms. The Streamable HTTP transport operates like a conventional API and allows the applying of standard network security controls. The STDIO transport, intended for local server processes, operates differently — the MCP client spawns an MCP server as a subprocess and communicates via standard input and output. This STDIO mechanism is central to the critical vulnerability identified by OX Security in April 2026.

The STDIO Remote Code Execution Flaw

OX Security researchers Moshe Siman Tov Bustan, Mustafa Naamnih, Nir Zadok, and Roni Bar documented what they described as "a by-design weakness in the MCP architecture that enables arbitrary remote code execution" [1]. As of April 2026, the flaw resided in all officially supported MCP language SDKs — Python, TypeScript, Java, and Rust — and stemmed from a single architectural decision: any process command passed to the MCP STDIO interface will execute on the host system regardless of whether the target binary initializes a valid MCP server.

The practical consequence is significant. An attacker who can influence what command an MCP client attempts to invoke as a local server can cause arbitrary code to execute on the developer's machine. The MCP client does not verify that the spawned process is a legitimate MCP server before executing it; it

simply spawns the specified command. This behavior propagates through every downstream implementation built on the official SDKs.

OX Security tested this at scale. Their team successfully poisoned nine out of eleven MCP registries with a test payload and confirmed arbitrary command execution on six live production platforms with paying customers [1]. OX Security reported that when they disclosed their findings to Anthropic, Anthropic characterized the behavior as "expected" within the protocol design and indicated that mitigation would be the responsibility of server operators rather than requiring protocol changes [1]. This response places the burden of mitigation on individual MCP server operators and end users rather than at the protocol level, a decision with significant supply chain implications given the ecosystem's scale.

The vulnerability's impact extends beyond individual developer machines. OX Security estimated the affected supply chain at more than 150 million downloads across the MCP SDKs, with exposure reaching up to 200,000 vulnerable installations across the affected supply chain [1][21]. For a protocol that has become infrastructure for production AI agent deployments, this represents an exceptionally broad blast radius.

Tool Poisoning, Shadowing, and Rug Pulls

Beyond the STDIO execution flaw, MCP's tool description architecture introduces a second family of attack vectors: attacks that manipulate the agent's behavior by corrupting the natural-language instructions it reads when interpreting tool capabilities.

Tool poisoning is the most direct of these techniques. An attacker who controls an MCP server can embed adversarial instructions in the tool's description field — a field the agent reads to understand what the tool does and how to use it. Because tool descriptions are natural language, they can contain behavioral directives that the agent will follow as if they were legitimate operational guidance. Invariant Labs documented this class of attack comprehensively in their MCP Security Notification [9], demonstrating that a single malicious tool description can redirect an agent's behavior across an entire session.

Tool shadowing represents a more sophisticated variant. Here, a malicious MCP server injects tool descriptions that modify the agent's behavior with respect to other, entirely legitimate MCP servers in the same session. The attacker does not need the agent to invoke their malicious tool directly; they simply need to influence the agent's interpretation of trusted tools [10]. This attack defeats the assumption that agents using only vetted tools are protected from malicious MCP server influence.

Rug pull attacks exploit the temporal dimension of trust. A malicious actor publishes a legitimate, functional MCP server that accumulates users and trust, then silently updates the server's tool definitions to include malicious behavioral instructions. By the time users encounter the updated behavior, they have already integrated the server into automated workflows and CI pipelines where per-invocation review is not occurring [10]. This attack pattern is especially dangerous in environments where MCP server versions are not pinned.

The OWASP MCP Top 10 formalizes these attack categories alongside additional vectors including authentication bypass, excessive permissions, and cross-server data leakage [11]. The taxonomy is new, but the underlying exploitation logic is consistent with other supply chain and dependency confusion attack patterns the industry has previously encountered — with the added dimension that the exploited component is actively reasoning about its environment and taking autonomous action.

Registry Ecosystem Risks

The MCP marketplace ecosystem amplifies each of these attack vectors. Several community-operated registries now index hundreds to thousands of MCP servers, and the installation friction is low — a configuration entry in the appropriate agent settings file is often sufficient to add a server to an agent's trusted tool set. OX Security found that across the registries they tested, only two of eleven successfully blocked their test poisoning payload before it was published [1].

This registry risk is structurally similar to the early history of npm and PyPI, ecosystems that took years to develop meaningful security scanning and package provenance infrastructure. The MCP registry ecosystem currently lacks code signing, behavioral sandboxing, automated malicious payload scanning, and meaningful publisher verification. Until these controls are established, organizations should treat any MCP server sourced outside of internally reviewed inventory as untrusted.

Repository Poisoning: Infecting the Developer Workspace

Configuration Files as Agent Instruction Channels

When a developer opens a repository in an AI coding agent, the agent reads the repository's contents — and in most implementations, specific configuration files function as first-class instruction channels. These files are designed to allow development teams to codify project-specific conventions, coding standards, and behavioral guidance for the agent. The security problem arises because these same files are version-controlled, repository-hosted artifacts that can be modified by anyone with write access to the repository, introduced through pull requests, or planted in compromised dependencies.

Pillar Security researchers disclosed the "Rules File Backdoor" attack in March 2025 [2]. Their research demonstrated that `.cursorrules` files in Cursor and `.github/copilot-instructions.md` files in GitHub Copilot can be used to inject adversarial behavioral instructions that the agent follows silently. What made this attack particularly difficult to detect was the technique's exploitation of hidden Unicode characters.

By embedding zero-width joiners, bidirectional text overrides, and other invisible Unicode characters within configuration file content, an attacker can create payload text that is semantically empty to a human reviewer performing a code review but fully legible to the AI model processing the file. The malicious instruction is present in the file's bytes; it is simply not rendered by standard text display tools. GitHub has since implemented a warning when files on github.com contain hidden Unicode text, but this control is advisory rather than blocking, and it operates only at the display layer rather than at the agent processing layer.

IDEsaster: Systematic Vulnerability Disclosure

The Pillar Security finding was not an isolated instance. Security researcher Ari Marzouk published the IDEsaster disclosure, a systematic review of security vulnerabilities across ten or more AI coding agent products, which identified more than thirty vulnerabilities and resulted in the assignment of 24 Common Vulnerabilities and Exposures (CVEs) [3]. The affected products include Cursor, Windsurf, GitHub Copilot, Roo Code, JetBrains Junie, Zed.dev, Cline, Claude Code, and OpenAI Codex CLI — effectively the entire landscape of current AI coding agent products used at enterprise scale.

The vulnerability class that produced the highest success rates was prompt injection via configuration files, where attackers hijack the agent's context through malicious rule files, MCP server configurations, deep links, or even file names. Research published in a September 2025 arXiv paper analyzing these attacks demonstrated a three-stage exploitation chain: prompt injection via a configuration artifact establishes behavioral control over the agent, which the agent then exercises through IDE tool integrations and base platform features, ultimately achieving code execution, credential access, or data exfiltration [7]. IDEsaster research on comparable attack chains reported success rates exceeding 84% against specific platforms [2] [3]; a separate January 2026 arXiv study found rates exceeding 85% under adaptive adversarial conditions [20].

Notably, different products exhibited materially different levels of resilience. Assessment results indicated that systems implementing mandatory tool confirmation dialogs and sandboxed MCP server execution demonstrated substantially lower attack success rates than systems that process configuration files without confirmation or egress controls [7]. This disparity in product-level security posture means that organizations should evaluate their chosen AI coding tools not only on capability but on their security architecture for handling untrusted context.

Attack Objectives and Real-World Impact

The objectives that prompt injection attacks via repository configuration files can achieve are broad. Research from a January 2026 paper evaluating attacks across 78 studies found documented attack chains achieving initial access and system discovery, credential theft from environment variables and filesystem

credential stores, data exfiltration to attacker-controlled infrastructure, modification of `.vscode/settings.json` to enable auto-approval of terminal commands without user interaction (CVE-2025-62222), and insertion of backdoored code into the repository being developed [20][3].

Discussions on Cursor's official community forum following the IDEsaster disclosure included user reports of AI coding agents running destructive commands during normal development sessions; while causation is difficult to establish from forum reports alone, the accounts are consistent with the configuration file poisoning vector described in IDEsaster [12]. The practical impact is that security incidents resulting from repository-poisoned AI coding agents may not be recognized as such: they may appear as developer error, system misconfiguration, or malware with unclear origin, because the proximate cause of the harmful action is the developer's own AI tooling.

Indirect Injection via Dependency Content

A related attack vector exploits the agent's practice of reading README files, code comments, and documentation from dependencies that a developer installs or references. An attacker who publishes a malicious package to a public registry can embed prompt injection payloads in that package's README.md, CONTRIBUTING.md, or inline code comments. When the developer's AI coding agent reads those files while assisting with dependency integration, it may execute the embedded instructions.

CSA's own research note on README instruction injection in AI coding agents, published in March 2026, documented this attack in detail and demonstrated proof-of-concept exploitation against several major agents [13]. The attack's effectiveness depends on the agent reading repository content automatically and on the content influencing the agent's subsequent tool use — both conditions that are satisfied by default in most current AI coding agent architectures.

The Miasma Worm: Supply Chain Meets AI Agent Auto-Execution

Campaign Overview

The Miasma worm represents, to the knowledge of the authors and available public threat intelligence, the first publicly documented malware campaign that uses AI coding agent configuration files as its primary propagation mechanism. The TeamPCP threat group, which threat intelligence researchers first identified in late 2025, launched the Miasma worm campaign beginning in May 2026. The campaign reached its highest-

profile phase on June 5, 2026, when GitHub disabled 73 repositories across four Microsoft GitHub organizations — Azure, Azure-Samples, Microsoft, and MicrosoftDocs — in an automated enforcement sweep that completed in 105 seconds [4][5].

The campaign operated in two distinct waves. The initial reconnaissance and credential-harvesting wave targeted individual developer machines by planting malicious configuration files in repositories likely to be opened by developers using AI coding agents. The second wave, known as the Hades Wave, extended the campaign's reach by pushing 37 malicious Python packages to PyPI across 19 package namespaces, using `.pth` startup hooks that silently stole credentials on every Python interpreter invocation [5].

Understanding Miasma's technical mechanism is essential for assessing its implications for the broader industry. The campaign's novelty lay not in exploiting zero-day software vulnerabilities — no CVEs were assigned across the entire campaign — but in weaponizing architectural behaviors that AI coding agents exhibit by design.

The Durabletask Incident: Technical Mechanism

The June 5 attack chain began with a compromised contributor credential that gave the attacker write access to the Azure/durabletask repository, a widely depended-upon library for building durable, long-running workflow applications in .NET and other environments. Using this credential, the attacker pushed a commit that planted multiple configuration files targeting different AI coding agent products: `.claude/` directory files targeting Claude Code, `.gemini/` configuration targeting Gemini CLI, `.cursor/` files targeting Cursor, and `.vscode/` settings targeting VS Code with Copilot [4][5].

Each configuration file was crafted to trigger automatic code execution the moment a developer opened the repository in the corresponding AI coding tool — not when they ran the code they were developing, but immediately upon opening the repository in their development environment. The payload executed a credential-harvesting script targeting Windows environment variables, with particular focus on Personal Access Tokens for GitHub and Azure, Azure service principal credentials, and Windows Hello authentication artifacts [5]. An independent technical analysis of the worm's propagation mechanism, including behavioral signatures and host indicators, was subsequently published by Security Joes under the campaign name "Shai-Hulud" [22].

The credential data collected from developer machines would grant the attacker access to additional repositories and CI/CD pipelines associated with the compromised developers. This propagation logic is what makes Miasma a worm in the traditional sense: successful exploitation of developer machines provides the credentials needed to plant the same configuration files in additional repositories, extending the campaign's reach through the trust relationships embedded in the software supply chain. The credential-harvesting payload extended to the machines of developers who open those repositories in AI coding tools configured to auto-process context files.

The Hades Wave: PyPI Component

Operating through the same compromised contributor account infrastructure that connected the Azure/durabletask incident to an earlier May 19 PyPI attack, the Hades Wave deployed 37 malicious Python wheels across 19 package names [5]. These packages used a less novel but complementary technique: Python's `.pth` file mechanism, which allows packages to execute arbitrary code during Python interpreter startup. Every Python invocation on a machine with one of the Hades Wave packages installed would silently transmit credential data to the attacker's command-and-control infrastructure.

Post-mortem analysis confirmed that the May 19 PyPI campaign and the June 5 repository injection both connected to the TeamPCP threat group through the secondary command-and-control domain `t.m-kosche[.]com`, a known TeamPCP infrastructure indicator [4]. This attribution connects the Miasma campaign to an actor with a history of multi-stage supply chain operations that combine traditional package registry exploitation with emerging AI-specific attack techniques.

Supply Chain Cascade: The Azure Functions Action Disruption

The most immediately consequential aspect of GitHub's enforcement response to Miasma was the disabling of `Azure/functions-action`, the official GitHub Action used by Azure Functions deployments. Because this Action was referenced by a large number of production CI/CD workflows across the ecosystem, its sudden unavailability blocked deployments at organizations that had no direct connection to any compromised repository or developer machine [4].

This cascade effect illustrates a systemic risk that the Miasma campaign exposed beyond the direct credential-harvesting objective: when a compromised repository is sufficiently central to the CI/CD ecosystem — whether through direct dependency or through widely referenced GitHub Actions — a defensive takedown creates its own disruption. Organizations that depend on publicly-hosted Actions without pinning specific commit SHAs discovered that the trust transference inherent in their CI/CD architecture extends not just to attacks but to their countermeasures.

Key Tactical Indicators

Analysts assessing their own environment for Miasma indicators or similar campaigns should attend to several tactical patterns identified across the campaign. Unexpected `.claude/`, `.gemini/`, `.cursor/`, or `.vscode/` directories in repository diffs are significant supply chain signals, as these directories are processed automatically by AI coding agents and are not commonly introduced in normal development commits. Commits that add configuration files in these directories without corresponding functional changes to application code warrant immediate review. Additionally, the presence of `.pth` files

in installed Python packages, particularly in packages installed recently or from accounts without established publication histories, should be treated as a high-priority indicator of potential credential-harvesting activity [4][5].

Agent Skills and the ToxicSkills Ecosystem

The Agent Skills Attack Surface

Parallel to the MCP marketplace, a separate ecosystem of AI agent skills has emerged. Agent skills — natural-language instruction packages defined in SKILL.md files and distributed through marketplaces such as ClawHub and skills.sh — allow users to extend AI coding agent capabilities with pre-packaged behavioral modules. A developer might install a "code review" skill, a "documentation generation" skill, or a "security audit" skill, each of which adds instructions to the agent's context that shape how it approaches the relevant task.

The security model of these skills is essentially nonexistent. Publishing a skill on ClawHub requires a SKILL.md Markdown file and a GitHub account that is one week old — no code signing, no security review, no behavioral analysis [14]. This is an unusually low barrier for software that will execute within a trusted AI agent context alongside credentials and source code.

ToxicSkills: Measuring the Threat

In February 2026, Snyk security researchers completed the first comprehensive security audit of the AI agent skills ecosystem, scanning 3,984 skills from ClawHub and skills.sh [14]. The findings established the scale of the threat empirically. Across the sample, 36.82% — or 1,467 skills — contained at least one security flaw. Of the full 3,984 skills audited, 13.4% (534 skills) contained critical-level security issues including malware distribution payloads, active prompt injection instructions, and exposed credential material.

Snyk's researchers identified ToxicSkills — malicious skills that appear functionally legitimate when reviewed visually but contain adversarial behavioral instructions that manifest only at runtime when an AI model processes them — as the paradigmatic attack type [14]. The most common ToxicSkills payload embedded instructions directing the agent to append environment variable values (including API keys, authentication tokens, and database credentials) as query parameters in HTTP requests made to attacker-controlled infrastructure. This technique is effective because it exploits the agent's legitimate practice of making HTTP requests as part of tool use; the exfiltration is embedded within normal operational behavior.

The analysis also found that 91% of confirmed malicious skills combined traditional malware patterns with prompt injection techniques, a convergence that defeats both AI safety mechanisms (which focus on the model's behavior) and conventional security scanners (which focus on code execution patterns) [14].

The ClawHavoc Campaign

Prior to the ToxicSkills audit, researchers at Koi Security and Antiy CERT documented the ClawHavoc campaign in January 2026: a coordinated deployment of 1,184 malicious skills distributed across 12 publisher accounts on ClawHub, all sharing the command-and-control IP address `91.92.242[.]30`, as reported in the Snyk ToxicSkills analysis [14]. The campaign targeted users of Claude Code and OpenClaw, representing the first documented coordinated malware campaign through the agent skills supply chain.

ClawHavoc demonstrated that the agent skills ecosystem has already transitioned from theoretical risk to active exploitation. The actors behind the campaign understood that skills installed by developers carry a higher level of implicit trust than arbitrary internet content — they are intentionally introduced into the agent's trusted context — making them an efficient delivery vehicle for credential-harvesting and behavioral manipulation payloads.

OWASP has responded by formalizing Malicious Skills as the AST01 (top-ranked) category in its Agentic Skills Top 10, reflecting the security community's assessment that skill supply chain attacks represent the leading threat to agent ecosystems in 2026 [8].

Threat Actor Perspective: The Economics of AI Agent Exploitation

Why AI Coding Agents Are High-Value Targets

Understanding the threat actor's perspective helps practitioners prioritize defensive investment. AI coding agents are attractive targets for several interrelated reasons. First, they routinely operate with access to credentials that have exceptional privilege: GitHub Personal Access Tokens with write access to repositories, cloud provider service principal credentials, database connection strings, API keys, and CI/CD pipeline secrets. These are the credentials an attacker needs to compromise a software supply chain.

Second, AI coding agents operate in a context of high developer trust and relatively low scrutiny of individual automated actions. A developer who has granted an agent terminal execution permissions generally does not review each command the agent runs; the agent's autonomy is the point. This creates a window in which a prompt-injected agent can take harmful actions without triggering immediate human review.

Third, the developer machine is a privileged position in the software supply chain. A compromised developer machine with access to source code repositories, build systems, and deployment infrastructure is the entry point for a supply chain attack that can affect every downstream consumer of the software that developer produces. The Miasma campaign exploited exactly this position — using compromised developer credentials obtained through AI agent exploitation to introduce malicious artifacts into trusted Microsoft repositories.

Multi-Stage Campaign Architecture

The Miasma campaign's architecture illustrates the multi-stage potential of AI agent exploitation. Stage one compromises developer credentials through MCP registry poisoning or repository configuration file injection. Stage two uses those credentials to introduce malicious configuration files into widely-used repositories, extending credential harvesting to the machines of developers who open those repositories in AI coding tools configured to auto-process context files. Stage three uses the expanded set of compromised credentials to push malicious packages to public registries, extending the campaign's reach to all systems that install those packages. This cascade — agent exploitation to supply chain compromise to broad ecosystem impact — can unfold without the assignment of a single CVE if the attack exploits design behaviors rather than memory corruption or authentication flaws.

This architectural pattern has implications for how organizations categorize and respond to AI agent security incidents. Traditional detection and response workflows that depend on CVE assignment, exploit signatures, and malicious code patterns may miss campaigns that operate entirely through behavioral manipulation of trusted agents and legitimate credentials obtained through those agents.

Defensive Countermeasures

Immediate Actions

Organizations actively deploying AI coding agents in environments with access to sensitive credentials or production repositories should prioritize the following controls. Agent execution environments require explicit scope limitation: coding agents should run with the minimum permissions needed for their intended task, which in most cases means read-only filesystem access by default, with write and execute permissions explicitly granted for specific workflows. Environment variables containing sensitive credentials should not be accessible to agent processes except where specifically necessary, and credential isolation through tooling such as secrets managers should be implemented to ensure that credentials are not exposed in the environment namespace that agents can read.

Repository configuration files that function as agent instruction channels require active governance. Security teams should add `.cursorrules`, `CLAUDE.md`, `.github/copilot-instructions.md`, `.claude/`, `.gemini/`, `.cursor/`, and `.vscode/` directories to code review checklists as security-relevant artifacts requiring elevated review scrutiny. Automated tooling to detect hidden Unicode characters in these files should be integrated into pre-commit hooks and CI pipeline validation steps. Unexpected introduction of these files in pull requests from external contributors, particularly in combination with changes to CI/CD configuration, should trigger a security review workflow rather than a standard code review.

MCP server inventories should be audited and rationalized. Organizations should maintain an approved inventory of MCP servers permitted in development environments, with version pinning to prevent silent rug pull updates. All MCP servers not sourced from internally reviewed implementations should be treated as untrusted by default. The STDIO transport mechanism, which carries the highest risk given the OX Security STDIO RCE finding, should be limited to servers that have been explicitly vetted by the security team.

Agent skill installations require a governance process equivalent to software package review. Given the ToxicSkills findings — where more than a third of publicly available skills contained security flaws — organizations should prohibit installation of skills from public marketplaces without a review process. Where possible, maintain an internal skill registry of vetted skills, and implement behavioral monitoring to detect agents exfiltrating data to unexpected external destinations during skill execution.

Short-Term Mitigations

Over a one-to-three-month horizon, organizations should implement more systematic controls. Network egress controls for agent processes are essential: AI coding agents should communicate through controlled egress paths where outbound connections can be logged and anomalous destinations flagged. This control is a prerequisite for detecting the credential exfiltration technique documented in both ToxicSkills and Miasma.

CI/CD pipeline hardening is a high-priority remediation given the supply chain cascade demonstrated by Miasma. GitHub Actions workflows should pin all external Action references to specific commit SHAs rather than mutable tags, preventing both Miasma-style worm propagation and rug pull attacks on Actions. Workflows should run with the minimum required GITHUB_TOKEN permissions, using the principle of least privilege consistently. Organizations whose workflows reference widely-used public Actions should assess the supply chain risk associated with those dependencies and implement mirroring strategies where appropriate.

Developer awareness and tooling education should address the new threat model that AI coding agents introduce. Developers using AI coding tools need to understand that the repositories they open, the skills they install, and the MCP servers they configure are all potential attack surfaces, not just the code they write

or the packages they install. Security training should be updated to include AI agent context hygiene as a standard practice alongside existing guidance on credential management and phishing awareness.

Strategic Considerations

Organizations at the leading edge of AI coding agent adoption should engage with emerging security standards and frameworks as those frameworks mature. Several efforts are establishing the technical vocabulary and control recommendations needed to systematically address this threat landscape, and early engagement provides both security value and the opportunity to shape standards to organizational context.

Behavioral monitoring and anomaly detection for AI agent sessions represents a maturing market with meaningful near-term utility. Products that log and analyze agent actions — tool invocations, file reads, network connections, shell commands — can provide the audit trail necessary to detect post-hoc exploitation and, with sufficient baseline development, flag behavioral anomalies in real time. This capability is a prerequisite for effective incident response in environments where agents take many automated actions per session.

For organizations with significant software supply chain exposure — those whose software is widely depended upon by downstream users — the Miasma scenario warrants specific tabletop exercises and incident response planning. The relevant scenario is not simply "an agent is exploited" but "an agent is exploited, credentials are obtained, and those credentials are used to introduce malicious artifacts into repositories whose downstream impact extends beyond the organization's immediate environment." This scenario requires coordination between the security team, the development organization, and communication channels to downstream consumers.

CSA Resource Alignment

The threat vectors described in this whitepaper map directly onto multiple Cloud Security Alliance frameworks, each of which provides relevant guidance and control vocabulary for organizational response.

MAESTRO (Multi-Agent Environment, Security, Threat, Risk, and Outcome) provides the primary threat modeling framework for agentic AI systems [15]. MAESTRO's seven-layer architecture — Foundation Models (L1), Data Operations (L2), Agent Frameworks (L3), Deployment and Infrastructure (L4), Evaluation and Observability (L5), Security and Compliance (L6), and Agent Ecosystem (L7) — maps precisely onto the attack surface described here. MCP vulnerabilities and tool poisoning operate at L3 (Agent Frameworks) and L7 (Agent Ecosystem). Repository and configuration file poisoning attacks operate at L2 (Data Operations), targeting the data the agent consumes. The Miasma worm's credential-harvesting

payload and supply chain cascade operate at L4 (Deployment and Infrastructure). MAESTRO's emphasis on cross-layer attack chaining is particularly relevant: the Miasma campaign is archetypal in its movement from L2/L3 exploitation to L4 impact.

The AI Controls Matrix (AICM) provides the control framework for operationalizing these mitigations [16]. Relevant AICM control domains include AI Supply Chain Security (covering agent skill and MCP server vetting), AI Identity and Access Management (covering credential scoping for agent processes), AI Incident Response (covering detection and response for agent exploitation scenarios), and AI Data Governance (covering what data agents are permitted to access and transmit). The AICM is a superset of the Cloud Controls Matrix (CCM) and should be the primary reference for organizations seeking to map AI coding agent controls to broader cloud security governance programs.

The Agentic Trust Framework (ATF), stewarded by the CSAI Foundation with attribution to founding author Josh Woodruff / MassiveScale.AI (CC BY 4.0), provides the governance architecture for autonomous agent deployments [17]. ATF's four-level maturity model — Intern (read-only, continuous oversight) through Junior, Senior, and Principal (autonomous within boundaries) — offers a direct framework for implementing the principle of least privilege for AI coding agents. In ATF terms, coding agents in production environments handling sensitive repositories should be provisioned at the Intern or Junior level as a default, with access to elevated capability (shell execution, credential access, upstream push permissions) governed explicitly and logged. The framework's concept of immediate demotion to Intern level upon a critical incident provides a useful operational model for isolation in response to suspected agent compromise.

The Agentic AI Red Teaming Guide, published by CSA's AI Organizational Responsibilities Working Group [18], provides the testing methodology appropriate for validating the controls described in this whitepaper. Organizations should use this guide to conduct structured red team exercises against their AI coding agent deployments, specifically targeting configuration file injection, MCP server poisoning, and credential exfiltration vectors.

STAR (Security Trust Assurance and Risk) provides the assessment and registry infrastructure for organizations seeking to evaluate AI service providers' security postures, including those offering AI coding agent products [19]. As the STAR registry extends its planned scope to cover agentic AI service characteristics, organizations should prepare to request and review STAR documentation from their AI coding agent vendors, with particular attention to declared behaviors around untrusted context processing, tool confirmation workflows, and egress controls.

CSA's **Zero Trust guidance** connects to the AI coding agent context through the principle that no entity — including an autonomous AI agent — should be implicitly trusted simply because it is an internal or authorized system. Zero Trust architecture applied to AI coding agents means explicit verification of agent

identity and permissions at each tool invocation, network segmentation that prevents agent processes from reaching credentials or systems they don't require for their current task, and continuous monitoring that treats agent behavior as subject to the same scrutiny as any other privileged process.

Conclusions

The threat vectors analyzed in this whitepaper — MCP auto-execution vulnerabilities, repository and configuration file poisoning, and the Miasma supply chain worm — are not speculative future risks. They are documented, actively exploited attack patterns with real-world casualties: 73 Microsoft repositories disabled, CI/CD pipelines blocked across the ecosystem, and credential material harvested from developer machines at affected organizations.

What unifies these attacks is a common structural insight: AI coding agents trust the content they read, and that trust is exploitable. The same properties that make these systems useful — their ability to parse context from any source and act on it autonomously — make them susceptible to adversarial context injection at every layer of the context hierarchy. MCP tool descriptions, repository configuration files, README documents, and agent skill packages are all data that agents consume as instructions, and adversaries have learned to plant instructions in each of these sources.

The appropriate organizational response combines technical controls, governance changes, and recognition that the threat model for software supply chain security has fundamentally expanded. Defending the software supply chain now requires defending the AI agents that developers use to build that software. This means auditing the context those agents consume, governing the permissions those agents exercise, and building the detection and response capability needed to identify when an agent has been compromised and is acting on behalf of an adversary rather than the developer who deployed it. This is a fundamental rethinking of trust models in the software development lifecycle — one that the documented incidents examined here show cannot wait.

The CSA AI Safety Initiative will continue to publish updated guidance as this threat landscape evolves. Security practitioners are encouraged to engage with the MAESTRO threat modeling framework, the AICM control catalog, and the Agentic Trust Framework to ensure that their organizational security posture keeps pace with the advancing capability and adversarial attention that AI coding agents are now receiving.

References

[1] OX Security. "The Mother of All AI Supply Chains: Critical, Systemic Vulnerability at the Core of Anthropic's MCP." OX Security Blog, April 2026.

[2] Pillar Security. "New Vulnerability in GitHub Copilot and Cursor: How Hackers Can Weaponize Code Agents Through Compromised Rule Files." Pillar Security Blog, March 2025.

[3] GBHackers. "Critical Vulnerabilities Found in GitHub Copilot, Gemini CLI, Claude, and Other AI Tools Affect Millions." GBHackers, 2025. (Secondary source; primary IDEsaster disclosure by researcher Ari Marzouk.)

[4] StepSecurity. "Miasma Worm Hits Microsoft Again: Azure Functions Action and 72 Other Repositories Disabled After Supply Chain Attack Targeting AI Coding Agents." StepSecurity Blog, June 2026.

[5] Phoenix Security. "Miasma Worm Reaches Microsoft Azure and PyPI: 73 Repositories Disabled, Hades Wave Drops 37 Malicious Python Wheels." Phoenix Security, June 2026.

[6] Authzed. "A Timeline of Model Context Protocol (MCP) Security Breaches." Authzed Blog, 2025–2026.

[7] arXiv. "'Your AI, My Shell': Demystifying Prompt Injection Attacks on Agentic AI Coding Editors." arXiv preprint arXiv:2509.22040, September 2025.

[8] OWASP Foundation. "AST01 – Malicious Skills." OWASP Agentic Skills Top 10, 2025–2026.

[9] Invariant Labs. "MCP Security Notification: Tool Poisoning Attacks." Invariant Labs Blog, April 2025.

[10] Natoma. "Understanding Model Context Protocol (MCP) Vulnerabilities: Rug Pull Attacks." Natoma Blog, 2025–2026.

[11] OWASP Foundation. "OWASP MCP Top 10." OWASP Foundation, 2025–2026.

[12] SafeDep. "Miasma Worm Targets AI Coding Agents via GitHub Repos." SafeDep, June 2026.

[13] Cloud Security Alliance Labs. "README Injection: Repository Files Hijacking AI Coding Assistants." CSA Labs, March 2026.

[14] Snyk. "Snyk Finds Prompt Injection in 36%, 1467 Malicious Payloads in a ToxicSkills Study of Agent Skills Supply Chain Compromise." Snyk Blog, February 2026.

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

[16] Cloud Security Alliance. "AI Controls Matrix (AICM)." CSA AI Safety Initiative, 2025–2026.

[17] Agentic Trust Framework. "Agentic Trust Framework v0.9.1." CSAI Foundation (stewarded from Josh Woodruff / MassiveScale.AI, CC BY 4.0), April 2026.

[18] Cloud Security Alliance. "Agentic AI Red Teaming Guide." CSA AI Organizational Responsibilities Working Group, 2025.

[19] Cloud Security Alliance. "STAR: Security Trust Assurance and Risk." CSA, 2025–2026.

[20] arXiv. "'Prompt Injection Attacks on Agentic Coding Assistants'." arXiv preprint arXiv:2601.17548, January 2026. (Meta-analysis synthesizing findings from 78 studies, 2021–2026.)

[21] OX Security. "MCP Supply Chain Advisory: RCE Vulnerabilities Across the AI Ecosystem." OX Security Blog, April 15, 2026.

[22] Security Joes. "Shai-Hulud: Miasma — When a Supply-Chain Worm Learned to Hijack AI Coding Agents." Security Joes Blog, 2026.

Additional Resources

The following sources provided supporting context during research and are included here for readers seeking additional background on related topics.

- Aim Security / arXiv. "EchoLeak: The First Real-World Zero-Click Prompt Injection Exploit in a Production LLM System." arXiv preprint arXiv:2509.10540, September 2025.
- Checkmarx. "11 Emerging AI Security Risks with MCP (Model Context Protocol)." Checkmarx Zero, November 2025.
- CSA Labs. "AI Coding Assistants as Attack Surface: Code, Skills, and Secrets." CSA Labs, April 2026.