Confused Deputy Attacks on Autonomous AI Agents

Prompt Injection, Privilege Escalation, and Autonomous Compromise Chains

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-23

Key Takeaways

The confused deputy problem — a classical access control vulnerability in which a privileged program is tricked by a less-privileged caller into misusing its authority — has re-emerged as a high-severity threat pattern in AI agent deployments, where credential delegation amplifies the impact of successful prompt injection beyond what earlier chatbot architectures permitted. When an AI agent holds broad credentials to act on an operator's behalf, any input channel that agent processes becomes a potential attack vector: an adversary who can write to an email inbox, a GitHub issue tracker, a retrieved document, or a tool output can inject instructions that the agent executes with the operator's full authority.

The February 2026 compromise of the Cline AI coding assistant illustrates the threat model concretely. A crafted GitHub issue title embedded a malicious instruction that triggered an authenticated Claude coding session to install an attacker-controlled package, which was subsequently distributed as an official update to approximately 4,000 developer machines [1][16]. The security firm grith.ai, which published the primary incident analysis, characterized it as "the supply chain equivalent of confused deputy": the developer authorized Cline to act on their behalf, and Cline — via compromise — delegated that authority to an agent the developer never evaluated, configured, or consented to [1][16]. (The findings of security researcher Jamieson O'Reilly of DVULN, and the grith.ai and adnanthekhan.com primary technical reports, are drawn upon throughout this note alongside the KrebsOnSecurity secondary account [1]; the DVULN primary advisory was not independently reviewed at time of publication.)

Three structural factors amplify this risk in current deployments. First, AI agents are designed to treat all content in their context window as potentially instructive, which eliminates the hard boundary between data and code that traditional security architectures rely on. Second, the same broad permissions that make agents useful also make the consequences of a successful injection severe and potentially irreversible. Third, the emergence of multi-agent architectures — where one AI agent orchestrates or communicates with others — creates propagation paths for confused deputy attacks that can traverse organizational boundaries without human oversight at each step.

Background

The confused deputy problem was first formally described by Norm Hardy in 1988 to explain a class of access control failures in capability-based operating systems [2]. The essential insight was that authority should be carried by the invoker of an action, not merely held in ambient form by the program performing the action. A compiler given access to a billing file for accounting purposes could be confused by a caller who named their output file identically to the billing file, causing the compiler to overwrite it on the caller's behalf — acting as a deputy that had been confused about whose interests it was serving.

The same structural dynamic is native to AI agents. An agent deployed for broad enterprise productivity — such as OpenClaw or similar agentic platforms designed for wide integration — may hold OAuth tokens for email and calendar services, API keys for cloud infrastructure, credentials for code repositories, and the ability to execute shell commands or invoke external APIs. These permissions are granted by an operator — typically a developer or enterprise administrator — with the expectation that the agent will use them only in service of legitimate, operator-sanctioned goals. The agent functions as a deputy: a trusted intermediary that acts on the operator's behalf within defined systems. The risk profile of any given deployment scales with the breadth of its credential grants.

The vulnerability arises from a property fundamental to language model–based agents: they process natural language instructions and natural language content through the same mechanism. When an agent reads a webpage, retrieves a document, processes an email, or receives a tool result, that content passes through the same inference pathway as explicit operator instructions. An attacker who can influence any of those content sources can attempt to inject instructions — "prompt injections" — that the agent interprets as legitimate directives and executes using its full authority. The agent, unable to verify the provenance of instructions embedded in content, becomes the confused deputy: a privileged actor manipulated into serving an adversary's goals while operating on credentials belonging to the operator.

The threat surface has expanded in proportion to the growing deployment of autonomous agents in enterprise environments. Where early agent deployments were narrowly scoped chatbots, current agents like Anthropic's Claude, Microsoft's Copilot, and the open-source OpenClaw framework are designed to integrate broadly across an organization's digital environment. OpenClaw, released in November 2025, is explicitly designed to "take the initiative on your behalf" by managing email and calendar, executing programs, browsing the web, and integrating with communication platforms including Discord, Signal, Teams, and WhatsApp [1]. The agent's value proposition — proactive, autonomous action across a wide integration surface — is precisely what makes it a high-value target for confused deputy attacks.

Security Analysis

The Anatomy of a Confused Deputy Attack Chain

For purposes of this analysis, a confused deputy attack on an AI agent can be understood as a four-stage chain. The attack begins with an injection vector: the adversary introduces malicious instructions into content the agent will process. This may be an email, a GitHub issue, a web page, a retrieved document, or a response from an external tool. Because agents are designed to act helpfully on instructions, the injected content does not need to bypass any authentication mechanism — it merely needs to reach the agent's context window through any channel the agent is configured to read.

The second stage is authority inheritance: the agent processes the injected instructions and begins executing them using its existing credentials. Unlike traditional code injection, there is no exploitation of a memory safety flaw or authentication bypass — the agent is simply doing what it was designed to do, applied to content it was not designed to treat as adversarial. The agent's own legitimate authority becomes the mechanism of exploitation.

The third stage involves action propagation, which may be immediate or may unfold over multiple inference steps. In the Cline incident, the injected instruction traversed three distinct execution environments: the GitHub issue was processed by an AI-powered issue triage workflow, which triggered a Claude coding session, which performed a package installation, which was incorporated into a software release pipeline [1][17]. Each step was individually plausible and individually authorized by some component of the development workflow. The confusion arose from the chain of authority delegation rather than any single unauthorized action.

The fourth stage, present in the Cline case and increasingly documented in research, is authority re-delegation: the compromised agent or its artifacts can themselves inject into subsequent systems. A malicious package installed by a compromised agent may contain further prompt injections targeting the agent that installed it or other agents operating in the same environment. A compromised AI coding assistant can introduce adversarial content into code repositories that will later be processed by other agents performing code review or automated testing. Multi-agent propagation of confused deputy attacks represents a particularly severe escalation path, as a single injection can traverse organizational boundaries and compound across multiple credential sets without triggering human review at each step. Recent research on Model Context Protocol (MCP) security has documented analogous patterns in which tool poisoning attacks traverse agent-to-server boundaries [3].

The OpenClaw Exposure Surface

The Cline incident, while severe, involved an active exploitation path. A parallel and arguably more widespread risk involves passive exposure of agent configuration and credentials through misconfigured administrative interfaces. Security researcher Jamieson O'Reilly of DVULN documented — as reported by KrebsOnSecurity [1] — that the web-based administrative interface for OpenClaw installations, when exposed to the internet without authentication controls, allows an external party to read the agent's complete configuration file, including every credential the agent holds: API keys, bot tokens, OAuth secrets, and signing keys. The ClawTrap red-teaming framework provides a systematic methodology for evaluating this and related OpenClaw exposure surfaces under realistic adversarial conditions [13].

The implications extend beyond credential theft. O'Reilly observed that an attacker with access to an exposed OpenClaw interface can inject messages into ongoing conversations, exfiltrate data through the agent's existing integrations in a manner that resembles normal traffic, and — critically — control what the human operator sees. As quoted in the secondary reporting [1]: "Because you control the agent's perception layer, you can manipulate what the human sees. Filter out certain messages. Modify responses before they're displayed." This capability undermines the human oversight model that most organizations implicitly rely on when deploying AI agents with significant operational authority: if the agent is compromised at the perception layer, the operator's ability to detect anomalous behavior is itself compromised.

A systematic search at the time of O'Reilly's report identified hundreds of OpenClaw installations with exposed administrative interfaces, according to the KrebsOnSecurity account of O'Reilly's findings [1]; the underlying DVULN primary advisory was not independently reviewed at time of this publication. This suggests that misconfiguration-induced confused deputy conditions are not isolated incidents but a pattern of deployment practice that organizations have not yet adapted to address.

Supply Chain Vectors and Skill Repository Risks

OpenClaw's ClawHub functions as a public repository of downloadable "skills" — extensions that allow the agent to integrate with and control additional applications. O'Reilly documented a supply chain attack path through ClawHub in which a malicious skill, once installed, could execute arbitrary instructions within the agent's authority context [1]. This pattern mirrors the software supply chain risks that have affected traditional software ecosystems, but with an additional AI-specific dimension: skill descriptions themselves may contain embedded prompt injections that influence agent behavior during installation or use.

Research published contemporaneously on arXiv presents a systematic threat taxonomy for Model Context Protocol systems — the protocol most widely used for AI agent tool integrations — cataloging 38 distinct threat categories including tool poisoning, server impersonation, and credential theft through malicious tool responses [3]. This taxonomy, designated MCP-38, documents that MCP servers can inject adversarial content into tool responses that the calling agent will process as trusted data, creating a confused deputy condition within the tool invocation pathway. This is structurally identical to the Cline supply chain attack: a trusted subsystem (an MCP tool server) delivers content that manipulates the invoking agent into taking unauthorized action.

Multi-Agent Authority Propagation

The security challenge compounds significantly in multi-agent deployments, where one AI agent orchestrates or invokes others. Most documented multi-agent architectures propagate authority implicitly — as illustrated by the agent control protocol research, which proposes explicit admission control precisely because current frameworks lack it [7]: when an orchestrator agent calls a sub-agent, the sub-agent may inherit some or all of the orchestrator's permissions, or the orchestrator may pass credentials directly as part of the invocation. Neither pattern provides adequate isolation for a confused deputy threat model.

Without additional safeguards such as message signing or out-of-band authorization verification, sub-agents receiving instructions from their orchestrator have no reliable mechanism to detect that the orchestrator's behavior has been manipulated. By default, current multi-agent frameworks provide no such safeguards — sub-agents receive instructions from their orchestrator through the same mechanism they receive legitimate instructions. The result is lateral authority propagation: a single injection into the orchestrator's context can affect every agent in the orchestration graph, each acting with its own independent set of credentials and permissions.

Recent research on access-controlled website interaction for agentic AI highlights the authorization gap in multi-agent delegation [4]. When an AI agent is delegated critical tasks — making purchases, modifying access controls, submitting forms — existing web authorization frameworks do not provide mechanisms for the website or service to verify that the delegating human actually authorized the specific action the agent is requesting. The agent presents human-derived credentials without the service being able to verify that the human's intent corresponds to the agent's action. This is the confused deputy problem extended to the service layer: the service is the deputy confused about whether the agent's request reflects genuine human authorization.

The Authority Scoping Gap

Underlying all of these attack patterns is a structural gap in how AI agent authority is currently defined and enforced. Unlike traditional software where permissions are granted at the function or API level and enforced by operating system or runtime mechanisms, AI agent permissions are typically granted as broad credential sets and enforced — if at all — by the agent's own safety training and instruction following. While major email APIs provide basic operation-level scoping — distinguishing read from send from delete — they do not support policy-level constraints such as restricting forwarding to internal domains or limiting access to emails matching a topic filter. The agent's authorization envelope is bounded by what its credentials permit operationally, not by the operator's semantic intent.

Research on post-training LLM agents for Linux privilege escalation demonstrates that AI agents can be trained or prompted to systematically identify and exploit privilege escalation opportunities — the same techniques that human penetration testers use, applied autonomously [5]. This research suggests that when an agent's authorization envelope includes operating system credentials or administrative access, confused deputy attacks could theoretically cascade into system-level compromise through automated privilege escalation chains that may outpace manual detection and response.

Recommendations

Immediate Actions

Organizations operating AI agents with production credentials should immediately audit the external exposure of agent administrative interfaces. Web-based agent management consoles should require authentication and should not be accessible from public networks without VPN or equivalent access controls. This addresses the passive exposure vector documented by O'Reilly, who identified hundreds of OpenClaw installations with publicly accessible administrative interfaces [1].

Agent credentials should be reviewed under a least-privilege lens with AI-specific considerations applied. Broad OAuth grants should be replaced with narrowly scoped grants where available. Where the underlying API does not support scoped credentials, the deployment should be reconsidered or isolated from high-risk data. Credentials used by AI agents should be distinct from credentials used by humans, enabling anomaly detection based on behavioral divergence without the noise of normal human variation.

Any CI/CD or automation workflow that processes user-submitted content — issue titles, pull request bodies, commit messages — and invokes an AI agent in response should be treated as a potential prompt injection surface. Inputs to such workflows should be sanitized to strip embedded instructions, and the AI agent's scope of action within the workflow should be limited to the minimum necessary for the stated purpose. In particular, agent-invoked code execution within CI/CD pipelines should be sandboxed and should not have access to production package signing or release infrastructure.

Short-Term Mitigations

Organizations should implement structured input validation before agent action execution — an approach analogous to prompt control-flow integrity [6][14]. Rather than allowing agents to act directly on instructions embedded in retrieved content, architectures should route all action-triggering instructions through a validation layer that checks whether the instruction source is an authorized principal. Content retrieved from external sources — web pages, emails, documents, tool responses — should be treated as untrusted data, not as instructions, unless it has been explicitly authorized through an out-of-band channel.

Agent admission control frameworks provide a complementary mitigation [7]. By requiring that each agent action be evaluated against a policy that specifies permitted action types, permitted targets, and required authorization levels, organizations can create a runtime enforcement layer that limits the blast radius of confused deputy attacks. Actions with irreversible consequences — software installation, mass email dispatch, credential modification, data deletion — should require explicit human confirmation regardless of the source of the instruction, enforced at the agent runtime level rather than relying on the agent's instruction following.

For organizations operating multi-agent architectures, inter-agent communication should be treated as a trust boundary. Sub-agents should not automatically inherit the authority of their orchestrator; each authority transfer in a multi-agent workflow should be an explicit grant, logged, and revocable. Agent-to-agent messages should be signed by the sending agent's identity, allowing receiving agents to verify the chain of authority and detect spoofed or manipulated orchestration messages.

Agentic deployments should be instrumented for behavioral anomaly detection at the action level. Logging of all tool calls, external API invocations, and file system operations — with alerts for deviation from established behavioral baselines — provides detection capability for ongoing confused deputy exploitation that is otherwise difficult to identify in real time. Research on systematic enumeration and coverage auditing of LLM agent tool call safety provides a methodology for assessing the completeness of anomaly detection coverage across agent action spaces [15]. The CSA Agentic AI Red Teaming Guide [8] offers step-by-step procedures for establishing behavioral baselines and designing test cases that validate detection coverage across the 12 primary agentic threat categories.

Strategic Considerations

At the architectural level, AI agent authority should be formally scoped at instantiation rather than implicitly determined by held credentials. Agents should declare their intended action space — the set of operations they are authorized to perform — and this declaration should be enforced by runtime controls external to the agent itself. This approach mirrors the principle that traditional software should declare its required permissions, with the operating system enforcing those declarations against actual system calls. Extending this model to AI agents requires new infrastructure — agent identity registries, capability enforcement layers, and audit mechanisms — that are the subject of active research [7] and are not yet available as production-ready components in mainstream agent deployment frameworks.

The accountability and traceability requirements that regulators are increasingly codifying — including provisions in the EU AI Act related to AI system oversight and logging, and provisions under consideration in U.S. AI governance frameworks [9] — are directly undermined by confused deputy attacks, which sever the chain of human-agent authorization without visible evidence. Organizations subject to these frameworks should treat confused deputy risk as a compliance issue, not only a security one.

Organizations should treat skill and plugin repositories for AI agent frameworks as third-party software supply chain risks, subject to the same vendor assessment processes applied to traditional software dependencies. The combination of broad agent authority and the potential for skill descriptions themselves to contain prompt injections creates a supply chain attack surface that is simultaneously novel and structurally familiar. The SLSA and NIST SSDF frameworks provide a foundation for supply chain security practices that can be adapted to AI skill repositories, though specific adaptations for the prompt injection threat vector remain an area of active development.

CSA Resource Alignment

This research note connects to several active areas within CSA's AI Safety Initiative and broader framework development.

The MAESTRO Agentic AI Threat Modeling Framework (Cloud Security Alliance, in active development) provides the foundational threat vocabulary for the attack patterns described in this note. MAESTRO's threat categories of Agent Authorization and Control Hijacking, Multi-Agent Exploitation, and Supply Chain and Dependency Attacks align with the confused deputy chain documented here. Organizations undertaking threat modeling for agentic deployments should use MAESTRO as the primary framework for enumerating risks and designing mitigations.

The CSA Agentic AI Red Teaming Guide [8] offers step-by-step testing procedures for permission escalation, control hijacking, and supply chain compromise in agentic systems — the three primary stages of a confused deputy attack chain as described in this note. Security teams should use this guide to validate that detection and prevention controls are effective against the specific attack patterns documented here before deploying agents with production credentials.

The AI Controls Matrix (AICM), which incorporates and extends the Cloud Controls Matrix for AI-integrated deployments, provides the control framework against which agentic security postures should be assessed. Relevant control domains include AI system access control, AI data governance, AI incident response, and supply chain security. Organizations should map their agentic deployment controls to the AICM and document gaps for remediation roadmaps.

CSA's Zero Trust guidance provides the architectural principle most directly applicable to multi-agent authority propagation: trust should never be implicit and should be continuously verified. Applied to agentic architectures, zero trust requires that each agent-to-agent authority transfer be explicitly granted, verified against policy, and logged — a principle that most documented multi-agent frameworks do not enforce by default [7], requiring organizations to implement explicit controls through architectural discipline.

The CSA blog series on agentic authorization published in March 2026 — including "Securing the Agentic Control Plane" [10], "Rethinking Authorization for the Age of Agentic AI" [11], and "AI Security: When Your Agent Crosses Multiple Independent Systems, Who Vouches for It?" [12] — develops the governance frameworks for agent identity and authority management that complement the technical controls described here.

References

[1] Brian Krebs, "How AI Assistants are Moving the Security Goalposts," KrebsOnSecurity, March 8, 2026. https://krebsonsecurity.com/2026/03/how-ai-assistants-are-moving-the-security-goalposts/

[2] Norm Hardy, "The Confused Deputy (or Why Capabilities Might Have Been Invented)," ACM SIGOPS Operating Systems Review, Vol. 22, No. 4, October 1988. DOI: 10.1145/54289.871709

[3] Yi Ting Shen, Kentaroh Toyoda, Alex Leung, "MCP-38: A Comprehensive Threat Taxonomy for Model Context Protocol Systems (v1.0)," arXiv:2603.18063, March 2026. https://arxiv.org/abs/2603.18063

[4] Sunyoung Kim, Hokeun Kim, "Access Controlled Website Interaction for Agentic AI with Delegated Critical Tasks," arXiv:2603.18197, March 2026. https://arxiv.org/abs/2603.18197

[5] Philipp Normann, Andreas Happe, Jürgen Cito, Daniel Arp, "Post-Training Local LLM Agents for Linux Privilege Escalation with Verifiable Rewards," arXiv:2603.17673, March 2026. https://arxiv.org/abs/2603.17673

[6] Md Takrim Ul Alam, Akif Islam, Mohd Ruhul Ameen, Abu Saleh Musa Miah, Jungpil Shin, "Prompt Control-Flow Integrity: A Priority-Aware Runtime Defense Against Prompt Injection in LLM Systems," arXiv:2603.18433, March 2026. https://arxiv.org/abs/2603.18433

[7] Marcelo Fernandez (TraslaIA), "Agent Control Protocol: Admission Control for Agent Actions," arXiv:2603.18829, March 2026. https://arxiv.org/abs/2603.18829

[8] Ken Huang et al., "Agentic AI Red Teaming Guide," Cloud Security Alliance, AI Organizational Responsibilities Working Group, 2025. https://cloudsecurityalliance.org/artifacts/agentic-ai-red-teaming-guide

[9] Shiliang Zhang, Sabita Maharjan, "Security, Privacy, and Agentic AI in a Regulatory View: From Definitions and Distinctions to Provisions and Reflections," arXiv:2603.18914, accepted at 2026 Governing Agentic AI Symposium, March 2026. https://arxiv.org/abs/2603.18914

[10] Cloud Security Alliance, "Securing the Agentic Control Plane: A New Foundation for Trust in AI," CSA Blog, March 20, 2026. https://cloudsecurityalliance.org/blog (direct post URL unavailable at time of publication)

[11] Cloud Security Alliance, "Rethinking Authorization for the Age of Agentic AI," CSA Blog, March 19, 2026. https://cloudsecurityalliance.org/blog (direct post URL unavailable at time of publication)

[12] Cloud Security Alliance, "AI Security: When Your Agent Crosses Multiple Independent Systems, Who Vouches for It?" CSA Blog, March 11, 2026. https://cloudsecurityalliance.org/blog (direct post URL unavailable at time of publication)

[13] Haochen Zhao, Shaoyang Cui, "ClawTrap: A MITM-Based Red-Teaming Framework for Real-World OpenClaw Security Evaluation," arXiv:2603.18762, March 2026. https://arxiv.org/abs/2603.18762

[14] Akshey Sigdel, Rista Baral, "Guardrails as Infrastructure: Policy-First Control for Tool-Orchestrated Workflows," arXiv:2603.18059, March 2026. https://arxiv.org/abs/2603.18059

[15] Xuan Chen, Lu Yan, Ruqi Zhang, Xiangyu Zhang, "Who Tests the Testers? Systematic Enumeration and Coverage Audit of LLM Agent Tool Call Safety," arXiv:2603.18245, March 2026. https://arxiv.org/abs/2603.18245

[16] grith.ai, "Clinejection: When Your AI Tool Installs Another," grith.ai Blog, February 2026. https://grith.ai/blog/clinejection-when-your-ai-tool-installs-another

[17] Adnan Khan, "Clinejection: Full Technical Analysis," adnanthekhan.com, February 2026. https://adnanthekhan.com/posts/clinejection/

