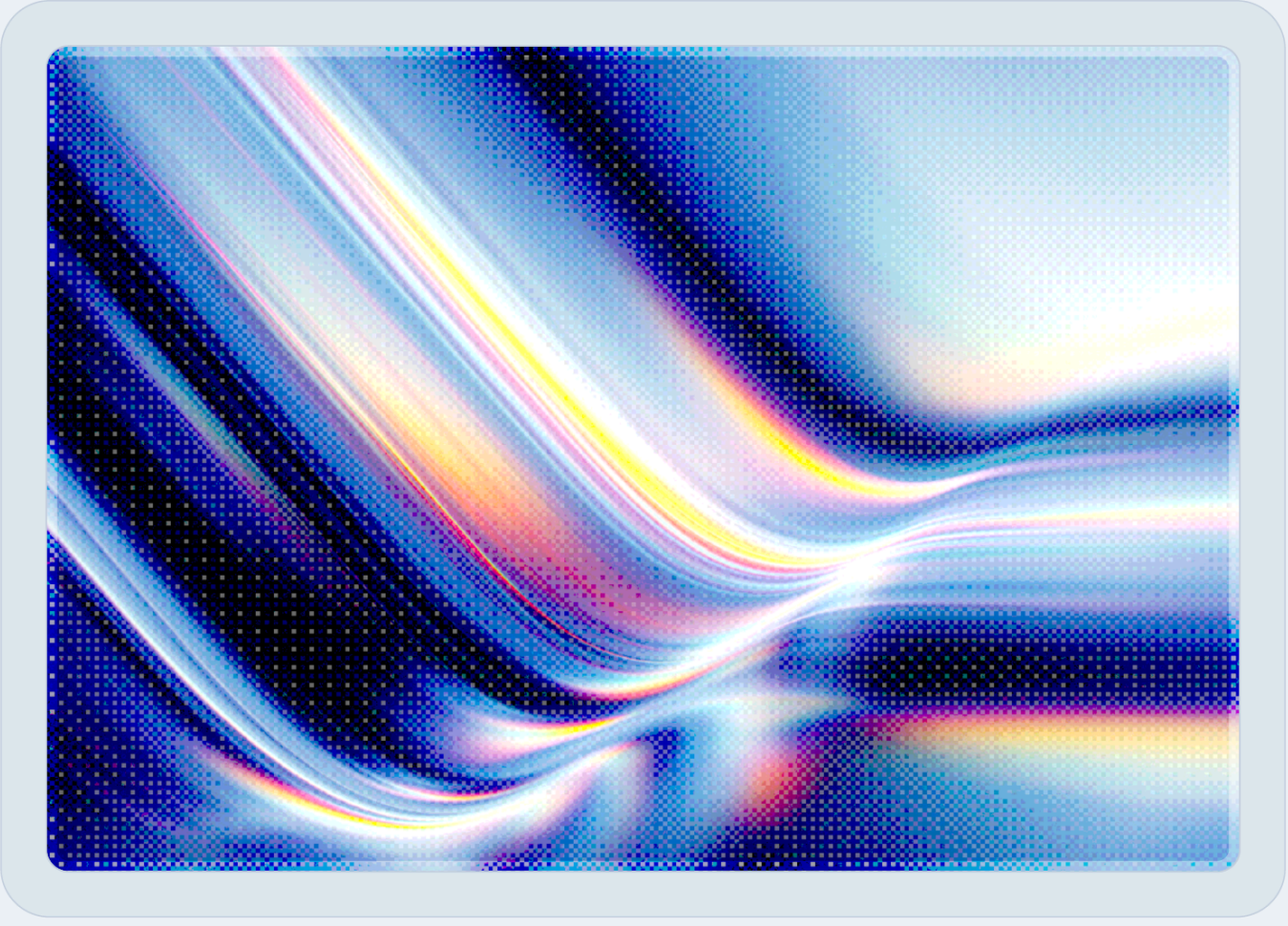
CSAI Foundation | Cloud Security Alliance

Confused Deputy Attacks on Autonomous AI Agents

Prompt Injection, Credential Delegation, and Agent Authority (Corrected Revision)

2026-11-03

AI-assisted Rapid Research



CSAI

cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

The confused deputy problem is a classical access control vulnerability in which a privileged program is tricked by a less-privileged caller into misusing its authority. It has re-emerged as a serious threat pattern in AI agent deployments, where credential delegation amplifies the impact of a successful prompt injection well 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, and the severity of the outcome scales with the breadth of the credentials and the reversibility of the actions available to the agent.

The January–February 2026 compromise of the Cline AI coding assistant illustrates the threat model concretely. An attacker placed a prompt injection in the title of a GitHub issue filed against the project. Cline's Claude-powered issue-triage workflow interpreted the title as an instruction and ran `npm install` against an attacker-controlled repository, whose install script poisoned the project's GitHub Actions cache. When the nightly release workflow later restored that cache, its npm publishing token was exfiltrated, and on February 17, 2026 the token was used to publish an unauthorized `cline@2.3.0` release whose only change was a post-install hook that silently installed the OpenClaw agent globally. The release was downloaded roughly 4,000 times before it was withdrawn about eight hours later [16][17]. The security firm grith.ai characterized the outcome 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 entirely separate agent, OpenClaw, that the developer never evaluated, configured, or consented to [16]. CSA has examined this class of prompt-injection-driven CI/CD compromise through AI coding agents in more depth elsewhere [19].

Three structural factors amplify this risk in current deployments. First, language-model agents process instructions and retrieved content through the same mechanism; although models are trained to prioritize operator instructions, that separation is learned rather than enforced, so there is no hard boundary between data and code of the kind traditional security architectures rely on. Second, the same broad permissions that make agents useful also make the consequences of a successful injection severe and, in some cases, irreversible. Third, the emergence of multi-agent architectures, in which one AI agent orchestrates or communicates with others, creates propagation paths for confused deputy attacks that can cross system boundaries without human oversight at each step.

Background

The confused deputy problem was named and described by Norm Hardy in 1988, in a paper recounting an access control failure at the timesharing company Tymshare in a conventional operating system whose protection structures Hardy compared to Unix — an ambient-authority, access-list design rather than a capability-based one [2]. A compiler installed in the system directory SYSX had been granted "home files license" so that it could write a statistics file in that directory, and the same directory also held the system's billing file. A user who supplied the billing file's name as the destination for the compiler's optional debugging output caused the compiler to overwrite the billing records, because the operating system checked only the compiler's license and not on whose behalf the request was being made. Hardy's diagnosis was that the compiler ran with authority from two sources — the authority its invoker lent it and its own home files license — and had no way to keep them apart or to express which one it intended to use for a given file. It was a deputy confused about whose interests it was serving. Hardy presented capabilities, which bundle the designation of an object with the authority to use it, as the design that prevents this class of failure; the paper's subtitle, "or why capabilities might have been invented," reflects that argument. The essential lesson is that authority should travel with the request that exercises it, rather than being held in ambient form by the program performing the action.

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 therefore 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, known as prompt injections, that the agent interprets as legitimate directives and executes using its full authority. Unable to verify the provenance of instructions embedded in content, the agent 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 deployments were narrowly scoped chatbots, current agentic products such as Claude Code, Microsoft 365 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 analysis that follows first sets out a general model of how confused deputy attacks unfold against AI agents, then examines the specific exposure surfaces that make them practical today: misconfigured administrative interfaces, skill and tool supply chains, and multi-agent orchestration. It closes by identifying the underlying gap in how agent authority is scoped, which explains why each of these surfaces is exploitable.

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 a channel the agent is configured to read.

The second stage is authority inheritance, in which the agent processes the injected instructions and begins executing them using its existing credentials. Unlike traditional code injection, this stage involves 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, so its own legitimate authority becomes the mechanism of exploitation.

The third stage involves action propagation, which may be immediate or may unfold over multiple steps. In the Cline incident, the injected instruction moved through four stages: the GitHub issue was processed by an AI-powered triage workflow, which installed an attacker-controlled package, whose install script poisoned the GitHub Actions cache, which in turn exposed the release workflow's publishing credentials

when the poisoned cache was restored [16][17]. The initial step — the triage agent installing a package because an issue title told it to — was a classic confused deputy action executed with the workflow's legitimate authority. It then enabled a conventional credential-theft step that no component of the workflow authorized. The incident thus combines an agent-specific failure of authority delegation with a familiar CI/CD weakness, and it was the combination that turned a prompt injection into a supply chain compromise.

The fourth stage, present in the Cline case, is authority re-delegation, in which the compromised agent or its artifacts 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. Similarly, 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 is a particularly consequential escalation path because a single injection can 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 cross agent-to-server boundaries [3][20].

The OpenClaw Exposure Surface

The findings in this subsection derive from the work of security researcher Jamieson O'Reilly of DVULN as reported by KrebsOnSecurity [1]; the DVULN primary advisory was not independently reviewed at the time of publication. Where the Cline incident involved an active exploitation path, a parallel and arguably more widespread risk involves passive exposure of agent configuration and credentials through misconfigured administrative interfaces. O'Reilly documented 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. Separately, the ClawTrap framework offers a man-in-the-middle, network-layer approach to adversarial evaluation of OpenClaw deployments under realistic conditions, which complements but does not directly address administrative-interface exposure [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.

According to the KrebsOnSecurity account of O'Reilly's findings, a cursory search at the time of the report identified hundreds of OpenClaw installations with exposed administrative interfaces [1]. The total population of internet-facing deployments was not reported, so the proportion that were exposed is unknown. Even so, the figure suggests that misconfiguration-induced confused deputy conditions are unlikely to be isolated incidents and instead reflect deployment practices 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," which are 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 the Model Context Protocol, a widely adopted protocol for AI agent tool integrations. Designated MCP-38, the taxonomy catalogs 38 distinct threat categories, including tool description poisoning, indirect prompt injection, parasitic tool chaining, and dynamic trust violations [3]. It 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; CSA's own analysis of MCP tool poisoning reaches the same conclusion [20]. This is structurally analogous to the initial stage of the Cline incident. In both cases, content arriving through a channel the agent treats as routine input manipulates it into exercising authority on someone else's behalf.

Multi-Agent Authority Propagation

The security challenge compounds in multi-agent deployments, where one AI agent orchestrates or invokes others. In many widely used multi-agent frameworks, authority transfer is implicit unless developers add controls such as per-agent tool allowlists or human approval gates. 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

isolates the sub-agent from a compromised orchestrator, which is the property a confused deputy threat model requires. Proposals such as the Agent Control Protocol, which introduces explicit admission control for agent actions, are motivated by gaps of this kind [7].

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. In many frameworks, sub-agents receive orchestrator instructions through the same channel as legitimate instructions, and the partial controls that do exist are typically opt-in rather than enforced by default. 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 delegation from a different angle [4]. When an AI agent is delegated critical tasks such as making purchases, modifying access controls, or 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, but the service cannot verify that the human's intent corresponds to the agent's action. This extends the confused deputy problem to the service layer, where the service itself becomes a 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. Agent credentials are typically API-scoped, as with traditional software, but the boundary that matters most — the operator's intent — is enforced only by the model's own behavior rather than by an external mechanism. OAuth scopes for major email APIs, for example, distinguish broad operation classes (read, send, modify) but cannot express content- or intent-level constraints such as "only messages about project X." Some tenant-wide administrative policies, such as blocking automatic external forwarding, can narrow the envelope, but they are coarse and are not designed around agent semantics. As a result, the agent's authorization envelope is bounded by what its credentials permit operationally, not by the operator's semantic intent.

This gap becomes more consequential when an agent's credentials reach the operating system. Research on post-training local LLM agents for Linux privilege escalation shows that agents explicitly tasked with the goal can learn to identify and exploit privilege escalation opportunities using techniques similar to those of human penetration testers [5]. That research concerns agents deliberately directed at privilege escalation rather than confused deputies. However, it suggests that if an injected instruction directed a

deputy agent holding operating system or administrative access to pursue such techniques, a confused deputy attack could theoretically cascade into system-level compromise faster than manual detection and response can follow.

Recommendations

The recommendations below are organized by time horizon. Immediate actions address the exposure surfaces documented above, short-term mitigations introduce runtime controls that limit the blast radius of a successful injection, and strategic considerations describe the architectural and governance changes needed to close the authority scoping gap.

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, as reported by KrebsOnSecurity, identified hundreds of OpenClaw installations with publicly accessible administrative interfaces [1].

Agent credentials should also be reviewed under a least-privilege lens with AI-specific considerations applied. Broad OAuth grants should be replaced with narrowly scoped grants where available, and 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, which enables anomaly detection based on behavioral divergence without the noise of normal human variation.

Any CI/CD or automation workflow that processes user-submitted content, such as issue titles, pull request bodies, or commit messages, and invokes an AI agent in response should be treated as a prompt injection surface. Untrusted fields should be passed to the agent as clearly delimited data rather than interpolated into its prompt, and the workflow should assume that injection will sometimes succeed. Accordingly, the agent's tools should be restricted to the minimum needed for the stated purpose; for triage, that typically means read and comment only, with no package installation or shell execution. The workflow should also run without access to caches, secrets, or credentials shared with release pipelines, and agent-invoked code execution within CI/CD pipelines should be sandboxed with no access to production package signing or release infrastructure. These are the controls that would have broken the Cline chain.

Short-Term Mitigations

Organizations should implement structured validation before agent actions are executed, an approach analogous to prompt control-flow integrity and to policy-first guardrails for tool-orchestrated workflows [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, including web pages, emails, documents, and tool responses, should be treated as untrusted data rather than 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. High-impact or hard-to-reverse actions, such as software installation, mass email dispatch, credential modification, and data deletion, should require explicit human confirmation regardless of the source of the instruction. That confirmation should be 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; instead, each authority transfer in a multi-agent workflow should be an explicit grant that is 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.

Finally, agentic deployments should be instrumented for behavioral anomaly detection at the action level. Logging 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 auditing whether safety evaluations cover an agent's full tool-call space, which can inform where runtime monitoring is most needed [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 its 12 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, meaning 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, including agent identity registries, capability enforcement layers, and audit mechanisms. These components are the subject of active research [7] and are not yet available as production-ready components in mainstream agent deployment frameworks.

Confused deputy attacks also complicate the traceability that the EU AI Act's record-keeping and human-oversight provisions (Articles 12 and 14) are intended to provide [22][9]. Because these attacks are carried out with legitimate credentials, they usually do leave records in API, tool-call, and CI logs; the Cline release, for instance, was flagged by automated monitoring shortly after publication [16]. The difficulty is attribution: logs will typically show the operator's credentials performing the action, obscuring that the instruction originated with an adversary. Organizations subject to such requirements should therefore treat confused deputy risk as a compliance issue, not only a security one.

Organizations should also 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. Taken together, these resources provide a threat vocabulary, testing procedures, a control framework, and an architectural principle that organizations can apply to the confused deputy patterns described above.

CSA's MAESTRO framework [18] models agentic systems as a seven-layer reference architecture, ranging from foundation models through data operations, agent frameworks, deployment and infrastructure, evaluation and observability, and security and compliance, to the agent ecosystem. The confused deputy chain described here spans several of these layers, most directly the Agent Frameworks, Deployment and Infrastructure, and Agent Ecosystem layers, which makes MAESTRO a useful structure for enumerating where injection, inheritance, propagation, and re-delegation can occur in a given deployment. The

complementary Agentic AI Red Teaming Guide [8] enumerates threat categories, including Agent Authorization and Control Hijacking, Multi-Agent Exploitation, and Supply Chain and Dependency Attacks, that map directly onto that chain.

The Red Teaming Guide [8] also offers step-by-step testing procedures for control hijacking, permission escalation, multi-agent exploitation, and supply chain compromise. Together, these procedures exercise each stage of the confused deputy chain described in this note: injection, authority inheritance, propagation, and re-delegation. Security teams should use the guide to validate that detection and prevention controls are effective against these attack patterns before deploying agents with production credentials.

The AI Controls Matrix (AICM) [21], 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 AICM domains include Identity and Access Management; Data Security and Privacy Lifecycle Management; Security Incident Management, E-Discovery and Cloud Forensics; Supply Chain Management, Transparency and Accountability; and Model 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. Because many multi-agent frameworks do not enforce this by default, organizations must implement explicit controls through architectural discipline.

CSA's 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 governance frameworks for agent identity and authority management that complement the technical controls described here. CSA's related research on AI coding agents as a supply chain node [19] and on MCP tool poisoning and the confused deputy problem [20] examines adjacent attack surfaces in greater depth.

Errata

Erratum (2026-11-03). This revision corrects several errors in the version of this note published 2026-03-23, which is superseded.

The first correction concerns the origin of the confused deputy problem in the Background section. The original note stated: "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]." That statement was inaccurate. Hardy's 1988 paper, "The Confused Deputy (or why capabilities might have been invented)," described a failure in a conventional ambient-authority, access-list operating system at Tymshare whose protection structures Hardy compared to Unix, in which a compiler holding "home files license" was tricked into overwriting a billing file; Hardy presented capabilities as the design that prevents the failure, not as its source. The original note also stated that the compiler had been given access to the billing file for accounting purposes, whereas in Hardy's account the license was granted so the compiler could write a statistics file, and the billing file was exposed only because it resided in the same directory. Reference [2] now carries the paper's full title, venue, pages, date, and a public URL.

The second correction concerns the Cline incident. The original note stated that the attacker-controlled package installed by the triage agent was itself distributed as an official update to approximately 4,000 developer machines, and that each step of the chain was individually authorized. According to the primary sources [16][17], the package install poisoned the project's GitHub Actions cache, which led to theft of the release workflow's publishing credentials; those stolen credentials were used to publish an unauthorized `cline@2.3.0` that silently installed OpenClaw, and that release was downloaded roughly 4,000 times. The publishing step was not authorized. The Key Takeaways and the Anatomy subsection have been corrected accordingly.

The third correction concerns CSA Resource Alignment, which attributed the threat categories Agent Authorization and Control Hijacking, Multi-Agent Exploitation, and Supply Chain and Dependency Attacks to MAESTRO. Those categories belong to the Agentic AI Red Teaming Guide [8]; MAESTRO is organized as a seven-layer reference architecture [18]. The section also now aligns the Red Teaming Guide's procedures with the note's four-stage chain rather than a "three-stage" chain.

In addition, this revision qualifies claims about multi-agent framework defaults, email API scoping, and regulatory traceability; replaces a recommendation to "sanitize" inputs to strip embedded instructions with delimiting and least-privilege controls consistent with the note's own analysis; corrects the descriptions of references [3], [5], [13], and [15]; updates the titles and dates of references [5], [16], and [17]; replaces blog-index URLs for references [10]–[12] with direct post URLs; removes an unsupported clause about U.S. governance frameworks; and adds references [18]–[22].

References

[1] Brian Krebs. "How AI Assistants are Moving the Security Goalposts." KrebsOnSecurity, March 8, 2026.

[2] Norm Hardy. "The Confused Deputy (or why capabilities might have been invented)." ACM SIGOPS Operating Systems Review, Vol. 22, No. 4, pp. 36–38, October 1988. DOI: 10.1145/54289.871709. Author-hosted full text: cap-lore.com.

[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.

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

[5] Philipp Normann, Andreas Happe, Jürgen Cito, Daniel Arp. "Towards Reliable Local Security Agents: Verifiable Post-Training for Linux Privilege Escalation." arXiv:2603.17673, March 2026.

[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.

[7] Marcelo Fernandez. "Agent Control Protocol: Admission Control for Agent Actions." arXiv:2603.18829, March 2026.

[8] Ken Huang et al. "Agentic AI Red Teaming Guide." Cloud Security Alliance, AI Organizational Responsibilities Working Group, 2025.

[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.

[10] Cloud Security Alliance. "Securing the Agentic Control Plane in 2026." CSA Blog, March 20, 2026.

[11] Cloud Security Alliance. "Rethinking Authorization for the Age of Agentic AI." CSA Blog, March 19, 2026.

[12] Cloud Security Alliance. "AI Security: When Your Agent Crosses Multiple Independent Systems, Who Vouches for It?" CSA Blog, March 11, 2026.

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

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

[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.

[16] grith team. "A GitHub Issue Title Compromised 4,000 Developer Machines." grith.ai Blog, March 5, 2026.

[17] Adnan Khan. "Clinejection — Compromising Cline's Production Releases just by Prompting an Issue Triager." adnanthekhan.com, February 9, 2026.

[18] Ken Huang. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 6, 2025.

[19] Cloud Security Alliance. "AI Coding Agents: An Unaudited Supply Chain Node." CSA Labs, 2026.

[20] Cloud Security Alliance. "MCP Security: Tool Poisoning and the Confused Deputy Problem." CSA Labs, May 16, 2026.

[21] Cloud Security Alliance. "AI Controls Matrix (AICM)." Cloud Security Alliance, 2025.

[22] European Parliament and Council of the European Union. "Regulation (EU) 2024/1689 (Artificial Intelligence Act)." Official Journal of the European Union, July 12, 2024.