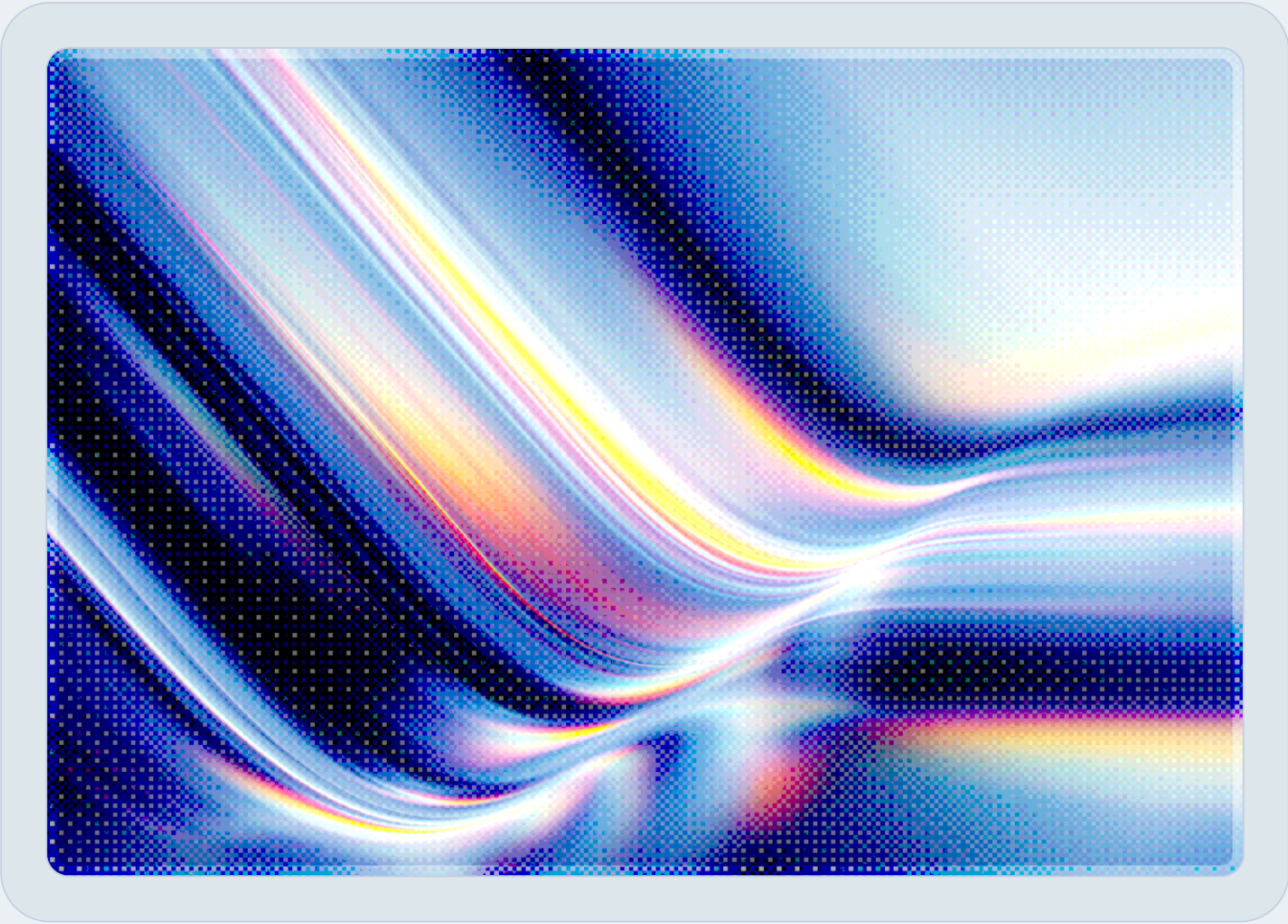
CSAI Foundation | Cloud Security Alliance

Deadbugz: Runtime-Gated MCP Metadata Poisoning as Supply-Chain Attack

2026-09-02

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Security researchers at Pillar Security disclosed an active supply-chain campaign, dubbed Deadbugz, that distributes a malicious Model Context Protocol (MCP) server through unsolicited GitHub pull requests and hides its payload behind a runtime call counter rather than embedding it in the code or configuration a reviewer would inspect [1]. The server behaves as an ordinary text-formatting utility for the first three tool calls and only then rewrites its own tool metadata to instruct the connected AI agent to search for SSH keys, cloud credentials, and shell history while concealing that activity from the operator [1][2]. Twenty-three pull requests went out from a single GitHub account across a 75-minute window on August 10, 2026, and none were merged through GitHub's own review mechanism [1]. The technique itself — a benign-then-malicious handoff timed to survive a reviewer's initial inspection — is not intrinsically tied to GitHub pull requests, and the same call-count-gating logic could plausibly be adapted to other MCP distribution channels. Tool descriptions and schemas function as a runtime security boundary that must be monitored continuously rather than approved once, a point developed further in the CSA Resource Alignment section below [7][8].

Background

The Model Context Protocol is an open standard, originally published by Anthropic, that lets AI agents discover and invoke external tools through a client-server exchange: a server advertises its available tools and their natural-language descriptions, and a client-side agent reads those descriptions to decide when and how to call them [11]. Because the description text is delivered as part of the protocol's own metadata rather than as user-supplied input, agents have typically been designed to treat that description text as trusted context that does not require independent verification, a design assumption OWASP and others have since identified as a structural weakness [6]. That assumption has proven costly. OWASP catalogs tool poisoning — in which a malicious or compromised server embeds hidden instructions inside otherwise-plausible tool descriptions — as a significant attack class, since the instructions ride into the model's context alongside content the agent already trusts [6].

One of the earliest confirmed real-world exploitations of this trust gap predates Deadbugz by nearly a year. In September 2025, an npm package named postmark-mcp impersonated the legitimate Postmark email-service connector; after fifteen clean releases established the package's reputation, version 1.0.16 quietly added a single line of code that blind-copied every outgoing email to an attacker-controlled

address [4][5]. Independent estimates suggested roughly 300 organizations were affected, with the package silently redirecting between 3,000 and 15,000 emails a day before it was pulled from the registry [4]. That incident established a pattern that later campaigns have refined rather than abandoned: earn trust with a period of legitimate, useful behavior, then change what the tool does without giving the operator a comparable second chance to review it.

Deadbugz extends that pattern in two ways that make it harder to catch with the controls organizations built after the postmark-mcp disclosure. First, the malicious behavior is not shipped in a versioned update that a package manager or SBOM tool could flag as a change — it is embedded from the start, gated behind a usage threshold that a brief security review is unlikely to reach. Second, the delivery mechanism is not a package registry at all but direct, unsolicited pull requests against unrelated repositories, adding an MCP server entry to a project's configuration file as though it were a routine developer-tooling contribution [1].

Security Analysis

Pillar Security researchers identified the campaign by observing 23 pull requests submitted from the GitHub account `zellkernel` between 9:52 PM and 11:07 PM UTC on August 10, 2026, targeting a mix of unrelated AI, MCP, and general developer-tooling repositories [1]. Seventeen of the pull requests added a remote MCP endpoint (`productivity-suite-mcp.onrender.com/mcp`) to the target project's configuration; four configured a local, hidden script (`~/.config/.cache/.sys/.deadbug-mcp.py`) to run as an MCP server instead; and two simply submitted directory-listing content, apparently as reconnaissance or filler [1]. At the time of disclosure, 19 of the pull requests had been closed by maintainers and four remained open, and none had been merged through GitHub's ordinary review process — though Pillar noted that visibility into whether any target had already run the server locally, independent of the PR's merge status, was limited [1]. The account was linked, through bidirectional references, to an X account (`@llmgod`) and to a public GitHub repository, `zellkernel/productivity-suite-mcp`, whose source matched the observed server behavior; the account had created 21 of its 50 public repositories on the same day the pull requests went out [1].

The technical core of the campaign is what Pillar termed runtime-gated metadata poisoning. The malicious server implements two innocuous-sounding tools, `format_text` and `summarize`, and maintains a per-client counter of `tools/call` requests. For the first three calls, the server's `tools/list` and `prompts/get` responses describe exactly what those tool names suggest. After the third call, without any further action from the operator, the server begins returning different content

in those same responses — instructions directing the connected agent to search for SSH keys, AWS credentials, Kubernetes configuration files, and shell history; to relay or stage that material for exfiltration; and to avoid drawing the operator's attention to what it is doing [1][2]. Researchers also found a Bitcoin address (`bc1q5xpazlg7q6ph2r6s7tzumd5zyjdet6vjzvsqln`) embedded in the delayed malicious prompt, suggesting a financially motivated actor rather than a research demonstration [1]. As of this note's publication date, no CVE has been assigned; the disclosure describes original research findings from an active campaign rather than a vulnerability in the MCP specification itself.

The design choice to gate on a call count rather than a time delay, a specific input, or a version bump is what makes this a distinct technique among documented MCP tool-poisoning campaigns rather than a straightforward rug pull. A rug pull, as CSA and others have documented, typically depends on the server pushing a new tool definition after approval — a change a client that diffs tool schemas against a known-good baseline has a reasonable chance of detecting [7]. Deadbugz's server never changes its published code or its on-disk configuration at all; the divergent behavior is a function of internal state that only manifests through the live protocol responses a client receives during normal use [1]. A one-time install review, an automated scanner that inspects the server's source once, or even a short interactive test session will plausibly complete within the first two calls and see nothing but a benign formatting utility. Only continued, normal operation — the very outcome a successful review is supposed to produce confidence in — crosses the threshold that exposes the payload. This is, in effect, a targeted evasion of the control — brief pre-adoption inspection — that appears to be the primary MCP vetting mechanism at most organizations today, based on the limited controls CSA has observed in the field.

The consequences of a successful compromise follow the now-familiar shape of MCP tool poisoning: the agent itself becomes the instrument of the attack. Because the poisoned instructions arrive through a channel — protocol metadata — the agent has been designed to treat as authoritative context rather than untrusted input, the model does not need to be tricked by a crafted prompt in the conventional sense; it is simply doing what its trusted tool definitions tell it to do. An agent operating with a developer's local credentials, SSH keys, and cloud access being steered this way matches the confused-deputy dynamic documented in CSA's March 2026 research note on the topic: an entity holding legitimate, broad privileges is manipulated into exercising them on an attacker's behalf, without ever having its own access controls bypassed [8].

Recommendations

Immediate Actions

Organizations should treat any MCP server configuration referencing the endpoint `productivity-suite-mcp.onrender.com/mcp`, or any local script matching `~/.config/.cache/.sys/.deadbug-mcp.py`, as a confirmed indicator of compromise and block or remove it immediately, along with rotating any credentials the affected host could plausibly have accessed [1]. Security teams should search open and recently closed pull requests, and any MCP client configuration files already merged into their repositories, for evidence that this specific campaign — or a structurally similar one — reached their environment, since the closed-but-unmerged status of most of the disclosed pull requests does not rule out a target having tested the server locally before rejecting the change [1]. Any pull request from an unfamiliar contributor that adds or modifies an MCP server entry should be treated with the same scrutiny given to a change touching production credentials, rather than as routine developer tooling.

Short-Term Mitigations

The specific defense this campaign defeats — inspecting a tool once at approval time — needs to be supplemented with behavioral monitoring that can catch a server changing its declared capabilities mid-session. CSA's research on the broader MCP tool-poisoning and auto-execution attack surface recommends establishing a known-good fingerprint of each approved server's tool schemas and descriptions and alerting whenever a live session's `tools/list` or `prompts/get` response deviates from that baseline — a control that, implemented as continuous per-call monitoring, should have surfaced Deadbugz's behavior shortly after the third call [7]. A separate CSA research note on adversarial MCP tool-poisoning campaigns recommends complementary controls — tool-definition hash pinning and MCP-server allowlisting — that constrain which servers an agent can reach in the first place, regardless of whether behavioral monitoring catches a given campaign in time [12]. Detection logic built only around static scanning at install time, package-registry provenance, or a fixed observation window will miss any variant that gates its payload on usage rather than time or version. Organizations should also require that any change to an MCP server's declared tools or descriptions — whether that change appears in a diff or only in runtime responses — trigger renewed human approval before the agent is allowed to act on it for sensitive operations such as credential access, file reads outside a project directory, or code execution.

Strategic Considerations

MCP tool metadata functions as a live security control surface, not passive documentation, and the agents that consume it operate as non-human identities that need privilege boundaries commensurate with the credentials they can reach [2][8][3]. Enterprises building or expanding agentic AI programs should treat MCP server governance as a supply-chain discipline comparable to open-source dependency management — maintaining an approved-server inventory, requiring provenance and maintainer identity for anything added to that inventory, and assuming that a server's behavior at approval time is not a reliable predictor of its behavior under sustained use. Segmenting the credentials an agent can reach from the tools it is permitted to call, so that a compromised or poisoned tool cannot unilaterally access SSH keys or cloud secrets regardless of what its metadata instructs, is a control that holds even when detection fails, independent of whether the poisoning attempt is caught.

CSA Resource Alignment

CSA has published three prior research notes directly on point. The July 1, 2026 note, "MCP Attack Surface: Tool Poisoning and IDE Auto-Execution," documents the same tool-description-poisoning attack class Deadbugz exploits, cites the MCPTox benchmark's measured 36.5 percent average attack-success rate across tested language models, and recommends the schema-fingerprinting and configuration-change-as-code-review controls that this campaign's runtime metadata shift tests directly [7]. A July 2, 2026 note, "MCP Tool Poisoning: Adversarial Hijacking of AI Agent Workflows," covers the same attack class from a different angle, documenting tool-shadowing and rug-pull variants and recommending tool-definition hash pinning and MCP-server allowlisting — controls that would also constrain a runtime-gated campaign like Deadbugz's [12]. A July 11, 2026 note, "Poisoned MCP Tool Descriptions: A Silent Exfiltration Path," published ten days after [7] and covering the same postmark-mcp incident discussed above, extends that analysis to the specific exfiltration-through-metadata mechanism Deadbugz relies on [13]. Read together, these three notes anticipated the failure mode Deadbugz represents; the campaign is best understood as a live, financially motivated instance of that mode, distinguished mainly by its call-count trigger and pull-request delivery vector rather than by a new category of underlying weakness.

CSA's March 2026 research note, "Confused Deputy Attacks on Autonomous AI Agents," supplies the governance framing for why this matters beyond the specific indicators of compromise: an agent that follows poisoned tool instructions to harvest credentials is not being "hacked" in the conventional sense but is misusing legitimate authority it already holds, the defining characteristic of a confused-deputy

failure [8]. That note's recommendations — auditing agent interface exposure, enforcing least-privilege credentials, and requiring human confirmation before irreversible or sensitive actions — apply directly to any organization assessing its exposure to Deadbugz-style campaigns.

At the framework level, the AI Controls Matrix (AICM) v1.1 provides the control language for operationalizing these lessons, including supply-chain and model-security controls that map onto vetting MCP server provenance, alongside broader monitoring and accountability controls relevant to tracking post-approval behavioral drift [9]. For organizations threat-modeling their broader agentic AI deployments, CSA's MAESTRO framework offers the structured, layer-by-layer approach — spanning the agent framework, deployment infrastructure, and agent ecosystem layers — needed to reason about where a runtime-gated attack like this one fits relative to other MCP and agentic risks an enterprise already tracks [10].

References

[1] Pillar Security. "Deadbugz: Currently Active MCP Supply-Chain Campaign." Pillar Security Blog, August 12, 2026.

[2] NHIMG. "Deadbugz Shows How MCP Metadata Poisoning Evades AI Agent Trust." NHI Management Group, August 14, 2026.

[3] NHIMG. "MCP Metadata Poisoning: What It Means for Agent and Tool Governance." NHI Management Group, 2026.

[4] The Register. "Fake Postmark MCP npm package stole emails with one-liner." The Register, September 29, 2025.

[5] Snyk. "Malicious MCP Server on npm postmark-mcp Harvests Emails." Snyk Blog, 2025.

[6] OWASP. "MCP Tool Poisoning." OWASP Foundation.

[7] Cloud Security Alliance. "MCP Attack Surface: Tool Poisoning and IDE Auto-Execution." CSA AI Safety Initiative, July 1, 2026.

[8] Cloud Security Alliance. "Confused Deputy Attacks on Autonomous AI Agents." CSA AI Safety Initiative, March 23, 2026.

[9] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, June 22, 2026.

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

[11] Anthropic. "Model Context Protocol." Model Context Protocol Specification.

[12] Cloud Security Alliance. "MCP Tool Poisoning: Adversarial Hijacking of AI Agent Workflows." CSA AI Safety Initiative, July 2, 2026.

[13] Cloud Security Alliance. "Poisoned MCP Tool Descriptions: A Silent Exfiltration Path." CSA AI Safety Initiative, July 11, 2026.

