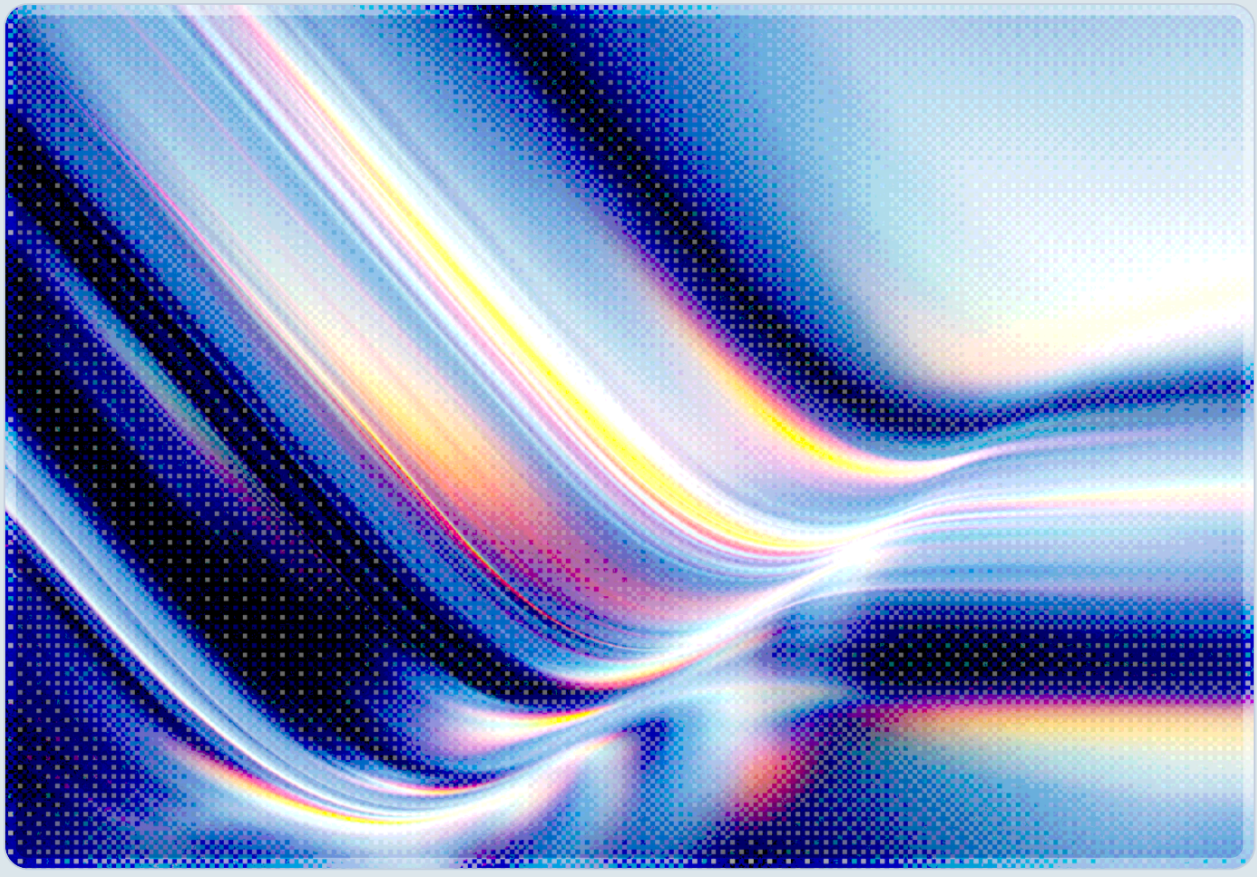
CSAI Foundation | Cloud Security Alliance

Gemini CLI CVSS 10: CI Supply Chain Code Execution

Workspace Trust Bypass Enables Unauthenticated RCE in Headless CI/CD Environments

2026-05-06

AI-assisted Rapid Research



CSAI

cloud security alliance®

Key Takeaways

- On April 24, 2026, Google published advisory GHSA-wpqr-6v78-jr5g disclosing a critical remote code execution vulnerability in `@google/gemini-cli`, rated CVSS 10.0 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H). The vulnerability affects all versions of the npm package before 0.39.1 (stable) and 0.40.0-preview.3 (preview), and the companion `google-github-actions/run-gemini-cli` GitHub Action before version 0.1.22 [1][2].

- The root cause is a workspace trust design flaw rather than a bug in the AI model itself. In headless (non-interactive) mode, Gemini CLI automatically trusted the current workspace folder, loading configuration files and environment variables from the `.gemini/` directory without explicit user approval or sandbox isolation. Any contributor who could introduce content into a repository's workspace — including via a pull request from an external fork — could plant a malicious `.gemini/` configuration that would execute arbitrary commands on the CI runner before the agent's sandbox ever initialized [1][3].

- A second, independently exploitable vulnerability existed in the tool's `--yolo` approval mode. When `--yolo` was active, the fine-grained tool restrictions defined in `~/.gemini/settings.json` were not enforced as expected. An allowlist intended to permit only `run_shell_command(echo)` would silently allow any shell command, because the unqualified `run_shell_command` entry acts as a wildcard in the policy engine [1][4]. Google corrected this in version 0.39.1 by reworking the policy enforcement path for `--yolo` mode [1].

- Successful exploitation of either vector provides full code execution on the host running the CI runner, with access to every secret, token, cloud credential, and source artifact the workflow could reach. The supply chain implications are severe: compromised CI runners serve as a natural lateral movement point into production deployment pipelines, cloud environments, and downstream software consumers [2][3].

- Organizations running Gemini CLI in CI pipelines that process external pull requests or untrusted repository content should treat this as requiring immediate remediation. Upgrade `@google/gemini-cli` to version 0.39.1 or 0.40.0-preview.3 and pin `google-`

`github-actions/run-gemini-cli` to version 0.1.22 or later. The GitHub Action defaults to the newest available CLI release unless explicitly pinned, meaning unpinned workflows were exposed from the moment of initial release [2][5].

Background

Google's Gemini CLI — distributed as `@google/gemini-cli` on npm — is an open-source terminal agent that brings Gemini's language model capabilities directly into developer workflows. Released publicly in 2025, it allows developers to query the model, automate coding tasks, analyze codebases, and integrate with external tools — all from the command line. The accompanying `google-github-actions/run-gemini-cli` GitHub Action extends this capability into automated CI/CD pipelines, enabling teams to trigger Gemini-driven analysis, code review, and task execution as part of their pull request and deployment workflows [2][6].

The integration of AI coding agents into CI/CD pipelines represents a structural shift in how development automation is constructed. Where a traditional CI pipeline executes deterministic scripts against clearly bounded inputs, an AI agent pipeline gives the agent discretion over what actions to take in response to its context — including the contents of the repository being processed. This discretion is powerful for legitimate uses, but it creates a novel attack surface: if an attacker can influence what the agent sees as its input or configuration, the agent itself can become the mechanism of exploitation. The Gemini CLI advisory provides an unusually complete illustration of this risk, combining workspace trust failure, tool policy bypass, and full host code execution in a single published advisory with a maximum CVSS score.

The vulnerability was discovered independently by Elad Meged at Novee Security and Dan Lisichkin at Pillar Security and disclosed to Google under coordinated vulnerability disclosure [1][3]. Meged described the significance of the finding in terms that go beyond the specific tool: "This vulnerability had nothing to do with prompt injection. It was an infrastructure-level issue, where attacker-controlled content was silently accepted as trusted configuration and executed before any sandbox was initialized" [3]. That framing points to the deeper problem — the gap between developers' intuitive model of how an AI coding tool behaves and the actual execution model that governs it in non-interactive environments. Concurrent with the Gemini CLI advisory, Novee Security also published CVE-2026-26268 affecting Cursor IDE, a high-severity vulnerability where a malicious Git hook embedded in a cloned repository could achieve out-of-sandbox code execution when the Cursor agent executed routine `git` operations

[7]. The proximity of these disclosures suggests an emerging research pattern targeting AI coding agent execution boundaries — organizations should treat the findings as potentially indicative of broader vulnerabilities in this class of tools rather than isolated incidents.

Security Analysis

Workspace Trust and the Headless Execution Model

The `.gemini/` directory is the mechanism by which Gemini CLI supports per-project configuration. In an interactive desktop session, the tool can surface trust prompts that allow the user to review and approve workspace-specific settings before they take effect. In a headless CI/CD environment — where there is no interactive terminal and no human present to approve prompts — this trust model fails to function as designed. Rather than refusing to load workspace configuration in the absence of user confirmation, vulnerable versions of Gemini CLI defaulted to treating the workspace as trusted, silently loading whatever `.gemini/` content it found [1][4].

The practical attack path is straightforward. An external contributor — or any account with the ability to open a pull request against the repository — introduces a `.gemini/` directory containing a crafted configuration file or `.env` file with malicious environment variable definitions. When a CI workflow using `google-github-actions/run-gemini-cli` processes that pull request, Gemini CLI initializes in headless mode, discovers the workspace configuration, and loads it without challenge. The malicious content executes as shell commands or environment variable injections on the runner host, before the agent's sandbox has been established and before the AI model processes a single token of user input [1][3][4]. The attacker never needs to craft a prompt or understand the model's behavior; the exploitation is entirely deterministic and infrastructure-level.

What makes the supply chain implications particularly acute is the execution context that CI runners typically carry. A workflow triggered on a pull request will commonly have access to repository secrets for code signing, cloud provider credentials for test environment access, package registry tokens for dependency resolution, and in some configurations, deployment credentials for staging environments. Code execution on the runner host — as opposed to within a sandboxed container — means the attacker can read all environment variables, interact with the host's network interfaces, and access any credentials mounted into the runner rather than just those explicitly passed to the agent. A single exploited CI run can therefore compromise secrets, cloud credentials, and deployment access that extend well beyond the repository being processed.

The Tool Allowlisting Bypass

The second vulnerability in the advisory is architecturally distinct but follows from a similar failure mode: an assumption in the policy enforcement logic that did not hold under `--yolo` mode. The `--yolo` flag is intended to allow trusted workflows to run without per-tool approval prompts, while still respecting fine-grained restrictions in `~/.gemini/settings.json` that limit which specific commands the shell tool may execute [1][4]. An operator might configure a policy that permits `run_shell_command(git status)` and `run_shell_command(echo)` but nothing else, intending to prevent the agent from invoking arbitrary shell commands.

In vulnerable versions, this restriction was not evaluated when `--yolo` was active. The policy engine treated `--yolo` as an unconditional approval bypass rather than an approval automation paired with continued policy enforcement. As a result, any workflow using `--yolo` with a shell-capable tool configuration was operating with full, unrestricted shell access regardless of the allowlist policy its operators believed was in place. The advisory's CVSS 10.0 score reflects the combination of both vulnerabilities — individually each is serious, but together they describe a scenario where no defensive configuration within the tool's own settings, short of disabling headless CI operation entirely, could prevent arbitrary code execution against untrusted repository content in a headless deployment [1][2].

Scope of Affected Deployments

Component	Vulnerable Versions	Patched Version
`@google/gemini-cli` (stable)	< 0.39.1	0.39.1
`@google/gemini-cli` (preview)	≤ 0.40.0-preview.2	0.40.0-preview.3
`google-github-actions/run-gemini-cli`	< 0.1.22	0.1.22

The exposure surface is broader than teams may initially recognize. The `run-gemini-cli` GitHub Action defaults to the newest available CLI version unless a version pin is specified. Organizations that added this action without explicit version pinning were exposed from the time they first deployed the

workflow through the April 24, 2026 patch date — and would have received the patch automatically on subsequent workflow runs only if they did not also pin the CLI version separately [2][5][11]. Teams that did pin should verify their pin references the patched version rather than an earlier release.

The Broader AI Coding Agent Attack Surface

The Gemini CLI vulnerability is illustrative of a structural pattern that extends across AI coding tools more broadly. Researchers at Novee Security and Pillar Security have described a "source-to-sink" attack model in which CI pipeline inputs — pull request titles, PR bodies, issue comments, code diffs, README files, and project configuration directories — function as attacker-controlled data that an AI agent reads and acts upon [3][4]. The legitimate value of AI coding agents in CI derives precisely from their ability to read all of this context and take responsive action; the attack surface is not separable from the feature.

Separate research by Aonan Guan demonstrated practical exploitation of this surface through prompt injection, achieving API key exfiltration from Gemini CLI, Claude Code, and GitHub Copilot Agent via maliciously crafted GitHub issue comments [8]. The Gemini CLI workspace trust bypass is distinct from prompt injection — it bypasses the model entirely — but the underlying exposure is the same: an AI agent sitting inside a CI pipeline holding CI execution privileges, reading from workspaces that external contributors can influence. The policy implication is that organizations integrating AI coding agents into CI must treat the agent's entire input surface as an adversarial attack surface, not just its model-facing prompt.

Recommendations

Immediate Actions

Organizations running any version of `@google/gemini-cli` before 0.39.1 in CI/CD pipelines, or using `google-github-actions/run-gemini-cli` before version 0.1.22, should treat those pipeline executions as potentially compromised if they processed any pull request or repository content from external or unreviewed sources. Upgrade all deployments to the patched versions immediately [1][2]. For the npm package, update `package.json` or `package-lock.json` to specify version 0.39.1 or later. For the GitHub Action, pin to `google-github-actions/run-gemini-cli@v0.1.22` or later in all workflow files. Do not rely on version-floating references that resolve to `latest`.

Following the upgrade, rotate all secrets accessible to any CI runner that executed a Gemini CLI workflow against unreviewed pull requests during the vulnerable window. This rotation must include GitHub Actions secrets referenced in the workflow environment, cloud provider access keys and service account credentials, package registry tokens, code signing keys, and any API keys or service credentials stored in repository secrets or CI platform variables [2][3]. The execution context of a typical CI runner means any secret the workflow could theoretically access should be treated as potentially exposed, regardless of whether logs show explicit evidence of exfiltration.

Audit GitHub Actions workflow files for version pins on all third-party actions, not only `run-gemini-cli`. Unpinned action references — those using a floating version tag or `@main` — present a related supply chain risk through action tag hijacking, a separate vulnerability class that the CI security community has documented extensively.

Short-Term Mitigations

For workflows that must continue running against untrusted pull request content, apply the principle of least privilege aggressively to the runner execution environment. Secrets that are not needed for untrusted PR workflows should not be mounted into those jobs at all — use separate job definitions with scoped permissions and secret access, triggered only after human review or in isolated merge-queue environments [4]. GitHub Actions supports this through the `pull_request_target` trigger with explicit review gates, and through OIDC-based short-lived credential dispensing for cloud providers rather than long-lived secret injection [4].

Review all Gemini CLI deployments for use of the `--yolo` flag and replace it with explicit, scoped tool allowlists that enumerate only the specific tools and commands required for the automation's purpose. The patched version enforces these allowlists correctly under `--yolo`, but the architectural preference should be to avoid blanket auto-approval modes in any CI context regardless of the version in use. The narrower the set of tool operations the agent is authorized to perform, the smaller the damage radius if a future vulnerability or prompt injection attack attempts to abuse those capabilities.

Configure workflows to use ephemeral runners for any job that processes external repository content. Ephemeral runners discard the entire runner environment after each job, preventing an attacker from using a compromised CI run to plant persistent backdoors on the runner host that would survive into subsequent workflow executions [4].

Strategic Considerations

The Gemini CLI advisory represents a category of risk that will accompany AI coding tools for as long as they are embedded in automated environments with execution privileges — not a product-specific defect that patches resolve permanently. The fundamental architectural challenge is that AI coding agents derive their value from broad context access and action capability, while security requires minimizing the trust granted to any component that processes untrusted input. These requirements are in structural tension, and the resolution requires organizational controls layered on top of product-level hardening.

Security teams evaluating AI coding tool integrations in CI should treat workspace configuration directories, model prompt contexts, and tool capability profiles as security-sensitive artifacts subject to the same review disciplines applied to other CI configuration. A `.gemini/` directory committed to a repository carries privilege implications comparable to a `.github/workflows/` directory — both shape what executes in the CI environment — and should be subject to the same review disciplines applied to workflow file changes. Automated policy enforcement that detects and alerts on changes to these directories would have given defenders earlier warning of a malicious payload introduced through a pull request.

The near-simultaneous disclosure of CVE-2026-26268 in Cursor IDE — also involving sandbox escape from an AI coding agent processing repository content, also discovered by Novee Security — indicates that this attack surface is actively being examined by security researchers [7]. Organizations should anticipate that additional vulnerabilities of this type will emerge across the ecosystem of AI coding tools, given the active research attention documented in concurrent disclosures, and build their defensive posture around the assumption of ongoing exposure rather than treating each individual advisory as isolated.

CSA Resource Alignment

The Gemini CLI supply chain vulnerability maps directly to CSA's MAESTRO agentic AI threat modeling framework, which addresses the specific risks that arise when AI agents operate with broad tool access in automated environments. MAESTRO's Layer 4 (Deployment and Infrastructure) addresses CI/CD pipeline contexts as a threat surface; the workspace trust bypass exemplifies exactly the risk MAESTRO describes — agents embedded in automated environments inheriting execution privileges from their deployment context [9]. Layer 3 (Agent Frameworks) similarly speaks to trust boundary failures of the kind exploited here, covering the compromised framework component scenarios and supply chain

vulnerabilities that malicious workspace configuration represents. CSA's February 2026 MAESTRO real-world application guidance addresses CI/CD pipeline threat modeling directly and provides a structured basis for evaluating Gemini CLI deployments [10].

CSA's AI Controls Matrix (AICM) treats software supply chain security and deployment pipeline integrity as cross-cutting concerns applicable across model provider, application provider, and infrastructure domains [12]. The AICM's controls for third-party component provenance verification, runtime dependency integrity, and the scoping of credentials available to automated workflows map directly to the defensive gaps the Gemini CLI advisory exposed. Organizations using AICM to govern their AI engineering practices should evaluate whether their CI pipeline configurations for AI coding tools meet the controls for least-privilege credential access and separation of trusted versus untrusted execution contexts.

The STAR (Security, Trust, Assurance, and Risk) program's cloud security questionnaire provides a mechanism for evaluating third-party AI tool vendors against supply chain security expectations. The Gemini CLI advisory, patched within two days of discovery and disclosed with full technical detail, represents a vendor response model that STAR assessors should acknowledge. However, the advisory also surfaces policy gaps that vendors cannot close through patching alone: whether AI coding tools should operate in headless CI environments against untrusted content, and what architectural constraints organizations should impose before they do, are governance decisions that STAR assessments can benchmark.

CSA's Zero Trust architecture guidance applies to the implicit trust model that the workspace bypass exploited. Treating the current working directory as inherently trusted in a headless CI environment — because a human developer would have reviewed it in an interactive session — is precisely the kind of implicit, context-dependent trust that Zero Trust principles reject. Translating Zero Trust to the AI coding tool layer means: verify workspace configuration explicitly before loading it, never grant an agent broader tool permissions than the current task requires, and treat all untrusted repository content as adversarial regardless of the contributor's account standing.

References

[1] GitHub Advisory Database. "Gemini CLI: Remote Code Execution via workspace trust and tool allowlisting bypasses (GHSA-wpqr-6v78-jr5g)." GitHub, April 24, 2026.

[2] SecurityWeek. "Critical Gemini CLI Flaw Enabled Host Code Execution, Supply Chain Attacks." SecurityWeek, April 2026.

[3] Elad Meged, Novee Security. "Google Gemini CLI CVSS 10.0 RCE Vulnerability: Critical Security Advisory." Novee Security, April 29, 2026.

[4] Penligent Security Research. "Gemini CLI RCE, Workspace Trust and the CI/CD Agent Attack Surface." Penligent, April 2026.

[5] The Register. "Google's fix for critical Gemini CLI bug might break your CI/CD pipelines." The Register, April 30, 2026.

[6] The Hacker News. "Google Fixes CVSS 10 Gemini CLI CI RCE and Cursor Flaws Enable Code Execution." The Hacker News, April 30, 2026.

[7] Assaf Levkovich, Novee Security. "CVE-2026-26268: How an AI Coding Agent Can Run Exploits in Cursor IDE." Novee Security, April 28, 2026.

[8] Aonan Guan. "Comment and Control: Prompt Injection to Credential Theft in Claude Code, Gemini CLI, and GitHub Copilot Agent." oddguan.com, April 2026.

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

[10] Cloud Security Alliance. "Applying MAESTRO to Real-World Agentic AI Threat Models: From Framework to CI/CD Pipeline." CSA Blog, February 11, 2026.

[11] CyberSecurityNews. "Google Gemini CLI Vulnerabilities Allow Attackers to Execute Commands on Host Systems." CyberSecurityNews, April 2026.

[12] Cloud Security Alliance. "AI Controls Matrix (AICM)." CSA, 2024.