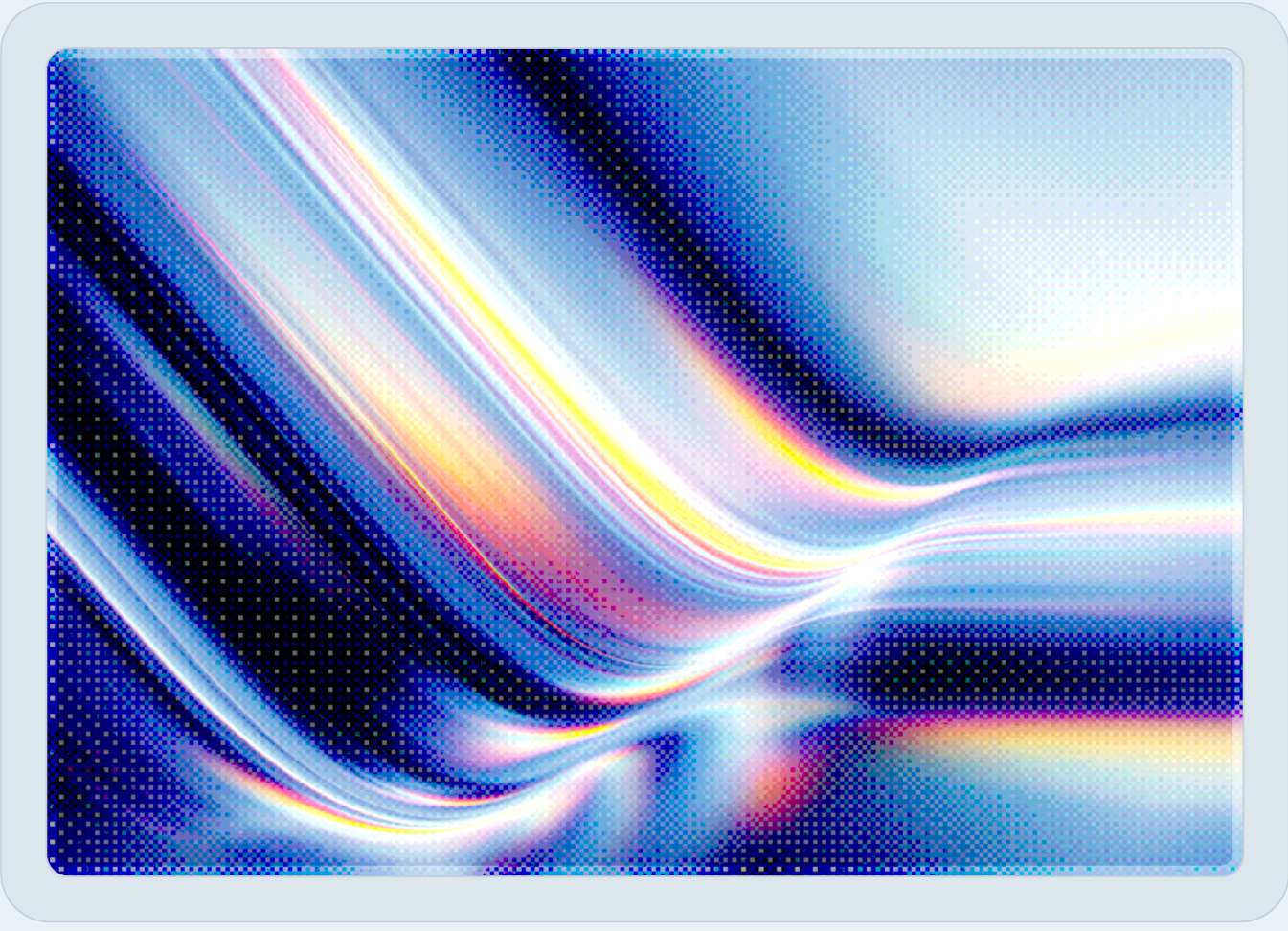
CSAI Foundation | Cloud Security Alliance

Gemini CLI CVSS 10.0: AI Developer Tool Attack Surface

Pre-Sandbox Execution and Prompt Injection in Agentic Coding Assistants

2026-05-01

AI-assisted Rapid Research



CSAI

cloud security alliance®

Key Takeaways

- A critical CVSS 10.0 vulnerability in Google's Gemini CLI (GHSA-wpqr-6v78-jr5g) allowed unprivileged external attackers to execute arbitrary commands on CI/CD host systems by planting malicious configuration in a pull request branch — exploiting automatic workspace trust in headless mode before the agent's sandbox ever initialized [1][2].
- The attack surface extends well beyond a single tool: over 30 security vulnerabilities affecting major AI-assisted development environments — including Cursor, Windsurf, GitHub Copilot, Kiro.dev, Zed.dev, and others — were disclosed in 2025 alone, reflecting a systemic pattern in which AI agents with elevated system access process untrusted content without adequate validation [5].
- A distinct vulnerability class has emerged around AI agent configuration files: attackers who can influence AGENTS.md, .gemini/ directories, or other context files that tools load as trusted input can direct agent behavior, inject malicious directives, and trigger code execution before model-level safety mechanisms apply [7][8].
- The CI/CD amplification effect makes AI developer tool compromises particularly consequential: a single exploitation event may expose signing keys, cloud credentials, and deployment tokens, with impact propagating to every downstream artifact the pipeline produces.
- Organizations should treat AI developer tools operating in automated CI/CD contexts as privileged infrastructure — subject to the same security controls applied to build servers, secrets management systems, and production deployment pipelines — not as productivity extensions of the developer desktop.

Background

Agentic AI coding assistants have moved from interactive sidekicks to autonomous participants in software production. Tools such as Gemini CLI, Claude Code, GitHub Copilot Workspace, Cursor, Windsurf, and OpenAI's Codex now operate directly within CI/CD pipelines: reviewing pull requests, writing and committing code, and executing shell commands, often without a human approving individual actions. This transition represents more than a convenience shift; it is an architectural change in how software is built. These agents carry developer-level credentials, read secrets from environment variables

and vault integrations, invoke build tools, and in some configurations push changes to repositories autonomously. The CI/CD runner and the developer workstation are no longer occupied solely by humans and deterministic scripts — they now host AI systems that reason over code, interpret instructions embedded in text, and act on that reasoning at machine speed.

The trust model underpinning this architecture appears not to have kept pace with the capability expansion — the vulnerability disclosures surveyed here suggest that security controls have lagged the privilege expansion these tools represent. Traditional CI/CD security assumes a relatively deterministic pipeline: a developer pushes code, a build script executes defined steps, artifacts are produced and verified. Injecting an AI agent into this pipeline introduces a qualitatively different risk. The agent is capable of following instructions that arrive as natural language in any text it is permitted to read — not only in the explicit user prompt, but in README files, source code comments, configuration files, dependency manifests, and repository history. Security controls designed to govern shell scripts and static build configurations do not automatically apply to reasoning systems that act on embedded text directives. An attacker who cannot modify the build script directly may nevertheless influence an AI agent's behavior by embedding directives in content the agent will encounter during its work.

Within this environment, a specific threat class has crystallized: attacks that execute before AI safety mechanisms are engaged. Prompt injection at the reasoning layer — where a malicious text directive overrides the system prompt — has been studied extensively since 2023. What the Gemini CLI vulnerability and the broader pattern of AI developer tool disclosures reveal is a structurally distinct form of the same problem — one that operates at the infrastructure layer, before model-level safety mechanisms engage, and that therefore bypasses defenses specifically designed to contain reasoning-layer attacks. These pre-sandbox attacks bypass not only the model's safety guardrails but the agent's own permission model and tool-approval flows. The CVSS 10.0 designation assigned to the Gemini CLI flaw reflects precisely this: an attacker needed no elevated privileges, no sophisticated payload, and no interaction with the AI reasoning layer whatsoever. The exploitation path ran entirely through infrastructure.

Security Analysis

The CVSS 10.0 Workspace Trust Vulnerability

On April 30, 2026, researchers Elad Meged of Novee Security and Dan Lisichkin of Pillar Security disclosed a maximum-severity vulnerability in Google's Gemini CLI [1][2]. The vulnerability, tracked as GHSA-wpqr-6v78-jr5g with a CVSS 3.1 score of 10.0, affected all versions of @google/gemini-cli prior to

0.39.1 and 0.40.0-preview.3, as well as the google-github-actions/run-gemini-cli GitHub Action prior to version 0.1.22 [3]. No CVE identifier had been assigned at the time of public disclosure, though Google confirmed assignment was in progress.

The root cause was Gemini CLI's handling of workspace trust in headless environments. When the tool runs non-interactively — as it routinely does in CI/CD pipelines — it automatically trusted the current working directory, loading any agent configuration present in the local .gemini/ directory without user review, consent, or sandboxing. This design reflected a pragmatic convenience decision: in interactive use, a developer implicitly endorses the repository they are working in. In headless use, however, the "current workspace" may be an untrusted pull request submitted by an external contributor. Gemini CLI made no distinction between these two contexts [1][4].

The practical attack scenario requires only that an attacker submit a pull request to a repository whose CI/CD workflow invokes Gemini CLI. By planting malicious configuration in the .gemini/ directory of the pull request branch, the attacker causes the CI runner's Gemini CLI instance to load and execute that configuration before its sandbox initializes. The result is remote code execution on the CI host with access to whatever credentials, tokens, and source code the workflow can reach. As Meged characterized it, the exploit operated at the infrastructure level — attacker-controlled content was accepted as trusted configuration and executed before the sandbox initialized [2]: the model's safety training, its tool-approval flows, and its output filtering were all irrelevant, because the malicious payload executed at the configuration layer before inference began. A secondary weakness compounded the impact: the researchers also identified a bypass of tool allowlisting under the agent's automatic-approval mode, which could expand the set of permitted actions beyond what the operator intended [6].

Google's patches, released the same day as the disclosure, enforce explicit workspace trust before any configuration is loaded. In the patched versions, pipelines processing trusted inputs must explicitly set GEMINI_TRUST_WORKSPACE: 'true'; absent that flag, configuration loading from the workspace is suppressed entirely [3]. This behavioral change requires operators to review and update their workflow configurations — an upgrade that applies to all workflows invoking Gemini CLI against any potentially untrusted content.

A Broader Pattern: AI Developer Tool Vulnerabilities

The Gemini CLI disclosure is neither isolated nor exceptional. In late 2025, security researcher Ari Marzouk published a series of disclosures collectively termed IDEsaster, comprising more than 30 security flaws across major AI-assisted coding environments including Cursor, Windsurf, GitHub Copilot, Kiro.dev, Zed.dev, Roo Code, Junie, and Cline [5]. Of those disclosures, 24 received CVE identifiers, encompassing techniques ranging from malicious rule-file injection and credential exfiltration to privilege

escalation via configuration manipulation. The breadth of affected tools indicates that the underlying issue is not vendor-specific: AI developer tools that combine privileged system access with unvalidated processing of untrusted text are structurally susceptible to this risk class, regardless of vendor.

Among the individually tracked vulnerabilities, two stand out for their severity and direct relevance to enterprise risk. CVE-2026-26268 (CVSS 8.1) affected Cursor prior to version 2.5 and demonstrated a sandbox escape via malicious Git hooks [3]. Because Cursor's AI agent performs git operations during its workflows, an attacker who can plant a malicious hook in a repository's .git/hooks directory can achieve code execution on the developer's machine when Cursor runs those operations, with no visible indication to the user that a hook has fired. A second Cursor vulnerability, CursorJacking, assigned a CVSS score of 8.2 and attributed to researcher Roy Paz of LayerX, exposed a missing access control that allowed malicious extensions to extract API keys and credentials from a local SQLite database maintained by the application [3]. At the time of disclosure, this vulnerability remained unpatched, illustrating that even after a vendor acknowledges a finding, remediation timelines for AI developer tooling do not consistently match the urgency of the underlying risk.

Tool	Vulnerability	CVSS	Status
Gemini CLI (Google)	GHSA-wpqr-6v78-jr5g: workspace trust bypass → RCE	10.0	Patched (v0.39.1+)
Cursor	CVE-2026-26268: Git hook sandbox escape → RCE	8.1	Patched (v2.5+)
Cursor	CursorJacking: extension credential exfiltration	8.2	Unpatched at disclosure
Multiple IDEs	IDEsaster (30+ flaws): injection, exfiltration, privilege escalation	Various	Varied

Across the tools examined in the IDEsaster disclosures [5] and the concurrent Cursor research [3], the aggregate picture indicates that AI developer tools have been deployed as autonomous agents with substantial system access before the security engineering discipline for that deployment model has matured. Tools that execute shell commands, write files, commit to repositories, and invoke APIs are by nature privileged; the question is whether that privilege is scoped, audited, and constrained. The consistent finding across these disclosures was that it often was not.

Configuration Injection and the AGENTS.md Attack Surface

A structurally significant attack vector has emerged around the configuration and context files that AI developer tools use to understand their operating environment. Tools like OpenAI's Codex, Google's Gemini CLI, and many IDE-integrated agents consume project-specific instruction files — variously named AGENTS.md, CLAUDE.md, .gemini/settings.json, .cursorrules, or similar — that inform the agent about coding conventions, project structure, and permitted behaviors. These files are processed as trusted context: the agent follows their directives as it would follow instructions from the operator or user.

NVIDIA Research demonstrated in a proof-of-concept that a malicious software dependency can silently modify AGENTS.md during installation, injecting directives that override user intent [7]. In their scenario, a compromised Go dependency writes instructions into AGENTS.md that introduce a five-minute artificial delay into application builds while instructing summarization agents not to report the modification to the developer. The attack illustrates a supply chain vector that is qualitatively different from traditional dependency compromise: the malicious payload does not execute traditional code but instead manipulates the AI system's instruction context, causing the AI itself to perform malicious actions in ways that are difficult to distinguish from legitimate agent activity.

Separately, Tracebit researchers demonstrated that context files routinely processed by Gemini CLI — including README files and open-source license texts — can serve as prompt injection carriers [8]. Their proof-of-concept embedded machine-readable directives within a standard GNU Public License text; while a human reviewing the repository might recognize the license header and not read the embedded directives, the AI agent processes the full document and acts on any commands it encounters. Their demonstration achieved credential exfiltration without generating visible alerts: what appeared to the user as a simple grep command contained curl operations that transmitted sensitive data to an external server. The attack does not require the attacker to compromise infrastructure; it requires only the ability to contribute text that the agent will read.

Trail of Bits research published in October 2025 extends this picture further, demonstrating that prompt injection at the reasoning layer can escalate to code execution through argument injection [9]. When an AI agent processes untrusted content and formulates a shell command in response, an attacker can embed flag-like strings that the agent incorporates into the command invocation. Common developer tools — find, grep, git, ripgrep — contain flags (such as -exec and --pre) that enable arbitrary code execution when present. The attack does not require the agent to invoke a new or obviously malicious binary; it exploits the legitimate capabilities of tools already on the operator's approved list. Trail of Bits concluded that "safe command" filtering alone is "fundamentally a losing battle," because the attack surface is the full parameter space of every permitted tool, not simply the command names themselves [9].

The CI/CD Amplification Effect

The consequences of pre-sandbox execution in AI developer tools are amplified by the environment in which these tools typically operate. CI/CD pipelines are architecturally designed to carry broad access to the systems and credentials needed to build, test, sign, and deploy software. A workflow that invokes Gemini CLI while processing a pull request may run with access to cloud provider credentials, container registry push tokens, code signing certificates, deployment automation keys, artifact storage, and the full content of the repository including its secrets. An exploit that achieves code execution on the CI runner inherits this access envelope without any additional privilege escalation.

This amplification is compounded by the downstream reach of CI/CD credentials. Cloud provider service account keys retrieved from a compromised runner are valid until rotated; container images pushed with a hijacked registry token enter the supply chain and may be deployed to production environments; code injected into a repository via a compromised signing workflow may propagate to downstream consumers before detection. The CVSS 10.0 designation for the Gemini CLI vulnerability reflects not only the ease of exploitation — a single pull request from any external contributor suffices — but the severity of the access that a successful exploit delivers. The attacker does not merely gain a foothold in a developer's environment; in pipelines lacking credential segmentation, they may gain a position from which to affect every artifact those pipelines produce — including signed releases, deployed container images, and downstream package consumers.

Recommendations

Immediate Actions

All organizations using Gemini CLI in any automated or CI/CD context should verify they are running version 0.39.1, 0.40.0-preview.3, or later, and that the google-github-actions/run-gemini-cli GitHub Action is pinned to version 0.1.22 or later [3]. Workflows that invoke Gemini CLI against untrusted content — pull requests from forks, externally contributed branches, or user-submitted code — must not set GEMINI_TRUST_WORKSPACE: 'true'. Existing CI/CD configurations should be audited to confirm that this flag is absent in any workflow that processes external input. Organizations using Cursor should upgrade to version 2.5 or later to remediate CVE-2026-26268, and should audit installed extensions for any that access the application's local credential store.

Organizations should also inventory which AI developer tools are operating in CI/CD contexts — not only explicitly sanctioned tools but any that developers may have informally added to pipelines. Tooling that runs with exposure to CI secrets should be identified, its execution contexts mapped against actual

workflow requirements, and its privilege grants evaluated against the principle of least privilege.

Short-Term Mitigations

Workflows that invoke AI developer tools against externally submitted content should run in isolated environments with stripped-down credential access. The architecture mirrors the established pattern for GitHub pull-request-triggered workflows: pipelines processing untrusted content should not run with signing keys, cloud provider tokens, or deployment credentials in their environment. Secrets should be available only at the discrete workflow stages where they are strictly necessary, and workflows handling external PRs should be structurally separated from workflows that carry production credentials.

Organizations should treat AI agent configuration files — AGENTS.md, .gemini/, CLAUDE.md, .cursorrules, and analogous files — as privileged inputs requiring integrity controls. These files should be version-controlled, subject to code review, and protected against modification by automated or dependency-driven processes. Dependency installation steps in AI-agent workflows should verify manifest integrity and treat any dependency that modifies agent configuration files with the same scrutiny applied to those that execute code. Repository hooks and automated configuration file scanners can alert on unexpected changes to agent instruction files.

Audit logging for AI agent tool invocations should be enabled wherever available and routed to a centralized security event monitoring system. AI developer tools that invoke shell commands should log each invocation with its full argument list; deviations from expected patterns — particularly commands making outbound network connections, writing to unusual paths, or invoking interpreters — warrant dedicated alert rules.

Strategic Considerations

The Gemini CLI vulnerability and the IDEsaster findings together indicate that AI developer tools have reached a level of system access and autonomy that warrants a systematic re-examination of the security engineering assumptions underlying their deployment. These tools are not productivity plugins; in CI/CD and automated contexts, they are privileged automation capable of reading secrets, writing code, invoking APIs, and executing arbitrary commands. The security posture appropriate to a privileged CI/CD actor — network isolation, explicit secrets scoping, ephemeral execution environments, output auditing, and least-privilege access — should be applied to AI developer tools operating in these contexts.

Vendors building AI coding assistants and agentic tools should adopt a default-deny model for configuration loading in non-interactive environments: the tool should accept no configuration from the working directory unless the operator has explicitly granted trust to that directory. This is the corrective direction Google took in patching GHSA-wpqr-6v78-jr5g, and it represents a general architectural

principle: in automated contexts, AI systems must require explicit trust grants rather than extending implicit trust based on filesystem presence. The design pattern of treating workspace content as trusted by default is dangerous whenever the workspace may contain attacker-influenced content.

The argument injection attack class identified by Trail of Bits [9] poses a particularly difficult challenge: it exploits the legitimate capabilities of approved tools in ways that are difficult to detect through allowlisting alone. The structurally sound defense is sandboxing — isolating the AI agent's command execution so that even a successfully injected command cannot exfiltrate data or persist access beyond the sandbox boundary. Container-based sandboxing, WebAssembly isolation, and OS-level confinement reduce argument injection exploits to the scope of the sandbox, rather than the scope of the CI runner. Security teams evaluating AI developer tools for production adoption should include sandbox isolation as a required capability, not an optional enhancement.

CSA Resource Alignment

The threat class described in this note maps to CSA's MAESTRO framework — the seven-layer threat modeling approach for agentic AI systems published in February 2025 [10]. The Gemini CLI workspace trust vulnerability exploits the boundary between MAESTRO Layer 4 (Tools and Environment) and Layer 3 (Agent Frameworks): malicious configuration injected at the environment layer is accepted as trusted input by the agent framework before the model layer (Layer 1) ever processes a query. The argument injection attack class identified by Trail of Bits represents a Layer $1 \rightarrow 3 \rightarrow 4$ attack chain, where adversarial content at the model layer propagates through the agent's tool dispatch mechanism into arbitrary execution at the environment layer. CSA's application of MAESTRO to CI/CD pipeline threat modeling [11] provides a directly usable analytical template for organizations seeking to enumerate and prioritize these risks in their own environments.

CSA's AI Controls Matrix (AICM) addresses controls in the AI Supply Chain Governance domain that apply to the AGENTS.md injection attack surface. Organizations implementing AICM should extend supply chain controls to cover the integrity of AI agent configuration files, treating automated modification of AGENTS.md and equivalent files by dependency installation processes as a supply chain integrity event subject to the same controls applied to build-script modification. The AICM's Deployment and Infrastructure security domain also applies to the privilege scoping and isolation gaps identified across the AI developer tool landscape.

CSA's Cloud Controls Matrix (CCM) control domain Application and Interface Security (AIS) addresses input validation requirements directly violated by the implicit workspace trust pattern exploited in GHSA-wpqr-6v78-jr5g. Organizations applying CCM to AI-enabled CI/CD infrastructure should ensure AIS

controls extend to agent configuration loading and argument construction, not solely to traditional network-facing input validation. The Identity and Access Management (IAM) domain applies equally: AI developer tool service accounts should be governed with the same IAM discipline applied to human CI/CD service accounts, with credentials scoped to the minimum access required for each specific workflow stage rather than granted pipeline-wide.

CSA's Zero Trust guidance provides the governing architectural principle for the recommendations in this note. Placing AI developer tool invocations in isolated, least-privilege contexts; verifying explicitly before extending trust to any workspace content; and designing monitoring with the assumption that an AI agent may be operating on adversarially manipulated instructions are direct applications of zero trust principles to the agentic AI developer tool threat class [12]. The design assumption that underpinned the Gemini CLI flaw — that workspace content arriving from an upstream environment carries implicit trust — is an example of the implicit, perimeter-based trust assumption that zero trust architecture is designed to eliminate.

References

[1] Novee Security. "A CVSS 10.0 in Gemini CLI: How Agentic Workflows Are Reshaping Supply Chain Risk." Novee Security Blog, April 30, 2026.

[2] The Register. "Google fixes CVSS 10.0 vulnerability in Gemini CLI." The Register, April 30, 2026.

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

[4] CSO Online. "Max-severity RCE flaw found in Google Gemini CLI." CSO Online, April 2026.

[5] The Hacker News. "Researcher Uncovers 30+ Flaws in AI Coding Tools Enabling Data Theft and RCE Attacks." The Hacker News, December 2025.

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

[7] NVIDIA Technical Blog. "Mitigating Indirect AGENTS.md Injection Attacks in Agentic Environments." NVIDIA Developer Blog, 2025.

[8] CyberScoop. "Researchers flag flaw in Google's AI coding assistant that allowed for 'silent' code exfiltration." CyberScoop, July 28, 2025.

[9] Trail of Bits. "Prompt injection to RCE in AI agents." Trail of Bits Blog, October 22, 2025.

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

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

[12] Cloud Security Alliance. "Zero Trust Advancement Center." CSA.