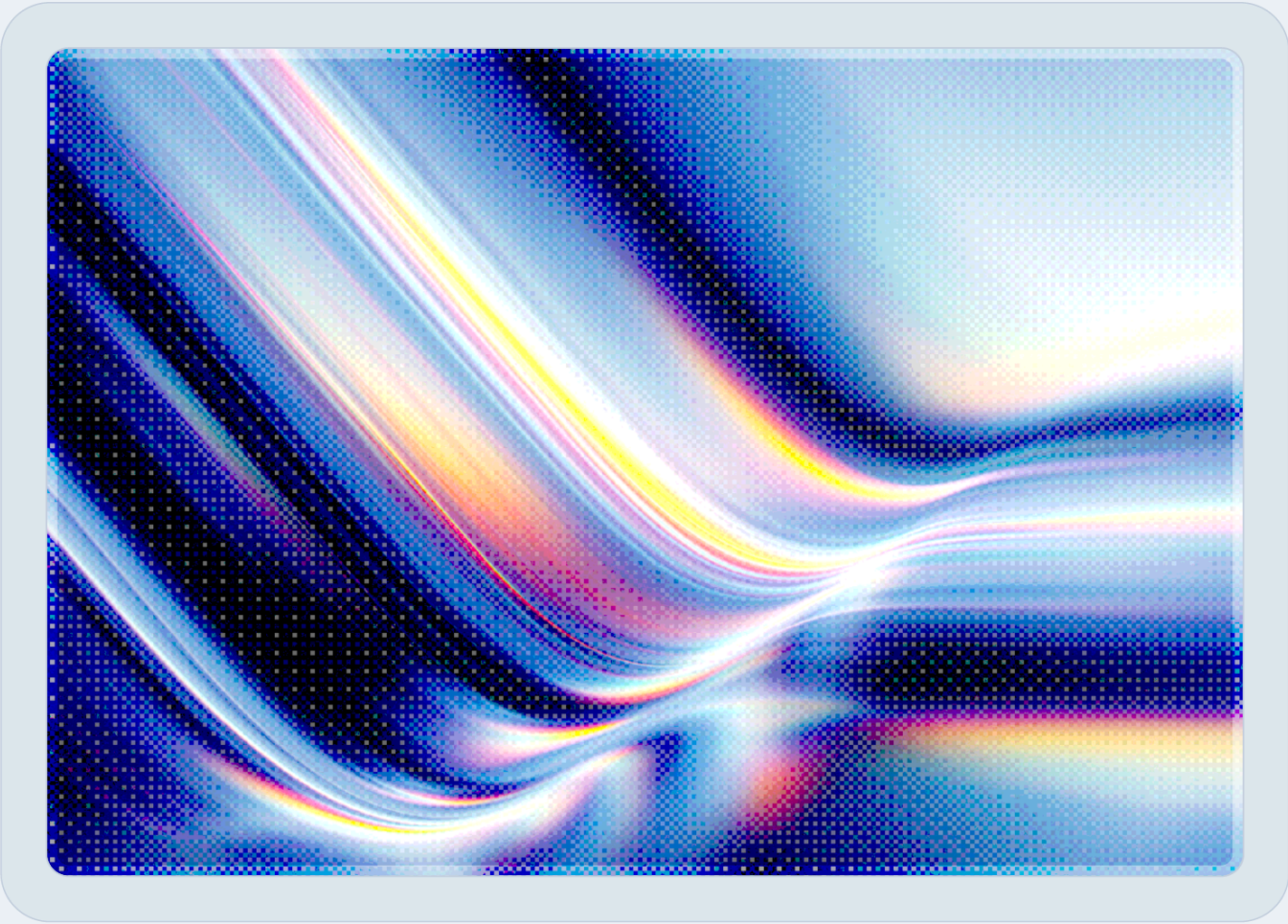
CSAI Foundation | Cloud Security Alliance

Cursor's Git.exe Zero-Day: Seven Months and No Patch

2026-07-15

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

A researcher at the AI security firm Mindgard disclosed on July 14, 2026, that Cursor, one of the most widely adopted AI-powered code editors, will automatically execute a malicious binary named `git.exe` when it is planted at the root of a repository opened on Windows, without any prompt, warning, or click from the user [1][2]. Mindgard reported the flaw privately on December 15, 2025, and after roughly seven months without a substantive fix or advisory from Cursor's maker, Anysphere, published full technical details, saying its goal was to give defenders the information needed to protect themselves absent a vendor fix [1]. The underlying weakness is not new: it follows the same binary-planting pattern that Microsoft's Git Credential Manager Core patched under CVE-2020-26233 in 2020, using an identical proof-of-concept technique of renaming the Windows Calculator to `git.exe` [2]. Independent researchers found the same class of flaw in GitHub Copilot CLI, Google Gemini CLI, and OpenAI Codex's desktop client, none of which had shipped a fix as of early June 2026, suggesting a pattern across the AI coding assistant market rather than an isolated Cursor defect [2]. Until Cursor ships a fix, the most reliable mitigations currently available are restricting execution from workspace directories through endpoint controls and treating any untrusted repository as hostile content that belongs in an isolated environment, not a developer's primary workstation [1][2].

Background

Cursor is a fork of Visual Studio Code that layers AI-assisted coding features, including autonomous multi-step "agent" workflows, onto a familiar editor interface. Its rapid adoption among professional developers has made it a high-value target: a compromise of a developer's Cursor session can expose source code repositories, cloud credentials, SSH keys, and any other secret reachable from that developer's machine. That concentration of access is what makes binary-planting vulnerabilities in developer tooling arguably more dangerous than similar flaws in ordinary consumer software.

The vulnerability Mindgard disclosed exploits how Cursor locates the Git binary it needs to inspect a repository when a project is opened. Rather than resolving `git` exclusively through a trusted system path, Cursor's search includes the workspace directory itself, and on Windows the operating system's default executable search order checks the current working directory before system-trusted locations [2]. When Cursor opens a project, it runs a command such as `git rev-parse --show-`

`toplevel` to determine the repository root; if an attacker has committed a file named `git.exe` into that root, Windows resolves and executes the attacker's binary instead of the legitimate Git client, and Cursor continues re-invoking it for as long as the project remains open [2]. Mindgard demonstrated the impact safely by renaming the Windows Calculator application to `git.exe` and placing it at the repository root: simply cloning the repository and opening it in Cursor was sufficient to launch the renamed Calculator with no dialog, warning, or user action required [1][2]. In a real attack, the planted binary would instead be a loader for a remote access tool, a credential harvester, or ransomware, executing silently with the logged-in user's full privileges.

Mindgard first reported the issue to Cursor's security contact on December 15, 2025 [1]. A subsequent submission through the company's HackerOne bug bounty program was delayed by what Mindgard described as an automation failure in the invitation process, pushing formal intake to mid-January 2026 [1]. Cursor's engineering team is reported to have closed early versions of the report as "Informative" or "out of scope," on the basis that the exploit path allegedly lacked a credible attack vector, despite Mindgard's working proof of concept [2]. Mindgard says it followed up multiple times between February and April 2026 without receiving a substantive technical response or remediation timeline [1]. A Cursor spokesperson told the trade publication Dark Reading on July 13, 2026, only that the company was "addressing this and will get back to Mindgard accordingly," a statement that arrived one day before Mindgard's public disclosure and carried no patch, advisory, or committed timeline [6].

Mindgard says the flaw was still present in the newest version of Cursor it tested, with confirmed reproduction in version 3.2.16 as of April 30, 2026, and continued reproduction in builds released as recently as July 10, 2026 [1][2]. Mindgard's writeup notes more than 197 new Cursor releases shipped across all channels in the roughly seven months since its initial report, and more than 70 additional releases shipped in the shorter window since its HackerOne submission alone [1]. Either figure illustrates the same underlying fact: Cursor's release cadence continued uninterrupted throughout the disclosure period without the flaw being addressed. As of this writing, no CVE identifier has been assigned to the issue [1][2].

Security Analysis

The technical root cause here is executable search-order hijacking, sometimes called binary planting: a program resolves an unqualified command name (`git`) by searching a list of directories, and if an attacker-controlled directory appears early enough in that search order, the attacker's file wins over the legitimate one. This class of bug has been understood and cataloged in Windows security guidance for well over a decade, and the specific "malicious git.exe at repository root" variant is not a novel discovery. Microsoft's own Git Credential Manager Core shipped an identical flaw, patched as CVE-2020-26233 in

2020, and Mindgard notes that the same Calculator-renamed-as-`git.exe` proof of concept that worked against that six-year-old bug reproduces cleanly against current Cursor builds [2]. That a well-documented, previously patched vulnerability class resurfaced in a widely deployed developer tool suggests that secure-defaults review for external binary resolution was either not applied to Cursor's Git integration or was not revisited as the product's agentic features expanded its autonomous file-system and process interactions.

In this analyst's assessment, three factors elevate this beyond a routine binary-planting bug: the zero-interaction execution path, the absence of any warning to the user, and the developer-focused target population. A conventional binary-planting exploit still typically requires a victim to run a specific command or open a particular file in a way that triggers the vulnerable search. Here, the act of opening a cloned repository in Cursor is itself sufficient, and Cursor repeats the execution automatically as long as the project stays open, giving an attacker multiple opportunities to succeed even if an endpoint control blocks the first attempt [2]. Because the attack requires nothing more than publishing a public repository and inducing a target to clone and open it, a practice increasingly associated with AI-assisted "vibe coding" and rapid prototyping workflows, this lowers the practical barrier to exploitation: no prior foothold, no social engineering beyond an enticing repository, and no interaction with security prompts a user might learn to distrust.

The disclosure timeline is itself a security-relevant data point. Seven months elapsed between private report and public disclosure, during which Cursor's team is reported to have dismissed the report's severity rather than engaging with the reproducible proof of concept, and the company's public acknowledgment arrived only after a journalist inquiry immediately preceded publication [1][2]. Whether this reflects under-resourced vulnerability triage, a genuine disagreement about exploitability, or a breakdown specific to Cursor's HackerOne intake process, the practical effect for defenders is the same: a working remote code execution path against a popular developer tool remained unpatched and largely unannounced for months, during which the only public information was the private knowledge held by Mindgard and Cursor.

The pattern extends beyond Cursor. Independent research cited alongside this disclosure found that GitHub Copilot CLI, Google Gemini CLI, and OpenAI Codex's desktop application each exhibited a comparable behavior, executing a workspace-supplied `git.exe` at startup or on folder open, and that as of June 4, 2026, none of the three vendors had shipped a fix: GitHub downgraded its version of the issue to "low" severity, Google had released no patch, and OpenAI classified its instance as "Not Applicable" [2]. This convergence across four major AI coding tools built by different vendors indicates a shared architectural assumption, that a workspace's contents can be trusted enough to search for supporting binaries within it, that has not kept pace with the reality that these tools are now routinely pointed at repositories of unknown provenance, including those an AI agent itself may be instructed to clone or fork during autonomous task execution.

Recommendations

Immediate Actions

Security teams supporting Windows-based developers should assume this vulnerability is exploitable in any environment running Cursor today, given the absence of a patch or committed timeline. Organizations with endpoint application control capability should deploy AppLocker or Windows Defender Application Control deny rules that block execution of binaries named `git.exe`, `npx.exe`, `node.exe`, and `where.exe` from paths under developer workspace roots, such as `%USERPROFILE%\source\repos\*` or equivalent project directories, since Cymulate's guidance, cited alongside Mindgard's findings, is explicit that these executable names "have no business" appearing inside a project root [2]. Security teams should also notify developers who use Cursor, Copilot CLI, Gemini CLI, or Codex desktop that opening any repository from an untrusted or unfamiliar source on Windows currently carries meaningful code execution risk, and should treat this guidance as applicable across all four affected tools rather than Cursor alone.

Short-Term Mitigations

Pending vendor patches, organizations should establish a practice of opening untrusted or newly cloned repositories inside an isolated, disposable environment, such as Windows Sandbox, a dedicated virtual machine, or a container with no access to production credentials, before ever opening them in a full-featured AI IDE on a primary workstation [1][2]. Repository intake processes, whether for open-source dependency evaluation, contractor code review, or onboarding forked projects, should include a lightweight scan for anomalous executables committed at a repository root, since a legitimately structured software project should never need to ship its own copy of `git.exe`, `node.exe`, or similar system tooling. Security teams should also confirm whether their software composition analysis or repository scanning tooling already flags unexpected binary files in source trees, and if not, treat this as a gap to close given how directly it maps to this exploitation pattern.

Strategic Considerations

This incident is best understood as evidence that AI-assisted developer tooling has inherited the security assumptions of traditional IDEs faster than it has inherited the threat model appropriate to autonomous, workspace-scoped execution. As agentic features increasingly grant these tools the ability to run commands, resolve dependencies, and interact with the file system on a developer's behalf with reduced human oversight per action, the cost of a stale assumption, such as trusting a workspace to supply its

own supporting binaries, rises accordingly, because there is less of a human in the loop to notice something is wrong before code executes. Organizations evaluating or standardizing on AI coding assistants should treat vendor responsiveness to vulnerability disclosure, not just feature capability, as a procurement criterion, and should ask prospective vendors directly how they resolve and trust binaries, plugins, or extensions referenced from within an opened workspace. Because this exact vulnerability class recurred nearly identically to a bug patched in 2020, organizations should also press AI coding tool vendors on whether their secure development lifecycle includes review against previously known vulnerability classes in the tools and libraries they build upon, rather than only original threat modeling of new AI-specific features.

CSA Resource Alignment

This incident maps closely to guidance CSA has already published on agentic AI system risk, even though the specific vulnerability is a conventional binary-planting bug rather than an AI model failure. CSA's Agentic AI Red Teaming Guide organizes agentic system risk into twelve threat categories, one of which is explicitly "Supply Chain and Dependency Attacks," the category that best describes how Cursor's workspace-scoped binary resolution allowed an attacker-supplied dependency artifact to substitute for a trusted system tool [3]. The guide's recommendation that organizations test how autonomous AI tools validate and resolve external dependencies before executing them is precisely the control gap this disclosure exposes, and security teams standing up red-teaming or pre-deployment evaluation programs for AI coding assistants should use that threat category as a starting checklist rather than treating this incident as a one-off IDE bug.

The Agentic Trust Framework (ATF), an open specification stewarded by the CSA-affiliated CSAI Foundation, applies Zero Trust principles to autonomous AI agents by defining what an agent is allowed to do based on demonstrated trustworthiness rather than default permission [4]. Its "Segmentation" pillar, which asks "where can you go?", and its identity and behavior verification elements extend directly to the binary-resolution failure described above: Cursor's agent-adjacent workflow effectively allowed a workspace-supplied file to substitute for a trusted system binary without any boundary check confirming the substitution was legitimate. Organizations building internal governance for AI coding tools can use ATF's four-level maturity model, which ties expanded autonomy to demonstrated trustworthiness and triggers automatic demotion after a critical incident, as a template for how much unsupervised file-system and process access to grant these tools by default.

Finally, CSA's AI Controls Matrix (AICM) v1.1 provides the broader governance backbone, with 247 control objectives spanning 18 domains that include application security and vulnerability management responsibilities applicable to organizations that build or deploy AI-enabled developer tools [5].

Application providers evaluated against the AICM are expected to demonstrate secure dependency resolution and timely vulnerability remediation, both of which this disclosure suggests were absent from Cursor's development lifecycle for at least seven months. Organizations conducting vendor risk assessments of AI coding assistant providers should incorporate AICM's application-provider control expectations into procurement questionnaires, using this incident as a concrete illustration of what inadequate coverage of those controls looks like in practice.

References

[1] Mindgard. "Cursor 0-Day: When Full Disclosure Becomes the Only Protection Left." Mindgard, July 14, 2026.

[2] The Hacker News. "Cursor Flaw Lets Malicious Cloned Repos Trigger Windows Code Execution." The Hacker News, July 2026.

[3] Cloud Security Alliance. "Agentic AI Red Teaming Guide." Cloud Security Alliance, 2025.

[4] Agentic Trust Framework. "Agentic Trust Framework (ATF) v0.9.1." CSAI Foundation, April 2026.

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

[6] Dark Reading. "Cursor IDE Auto-Executes Malicious Code in Poisoned Repos." Dark Reading, July 2026.

