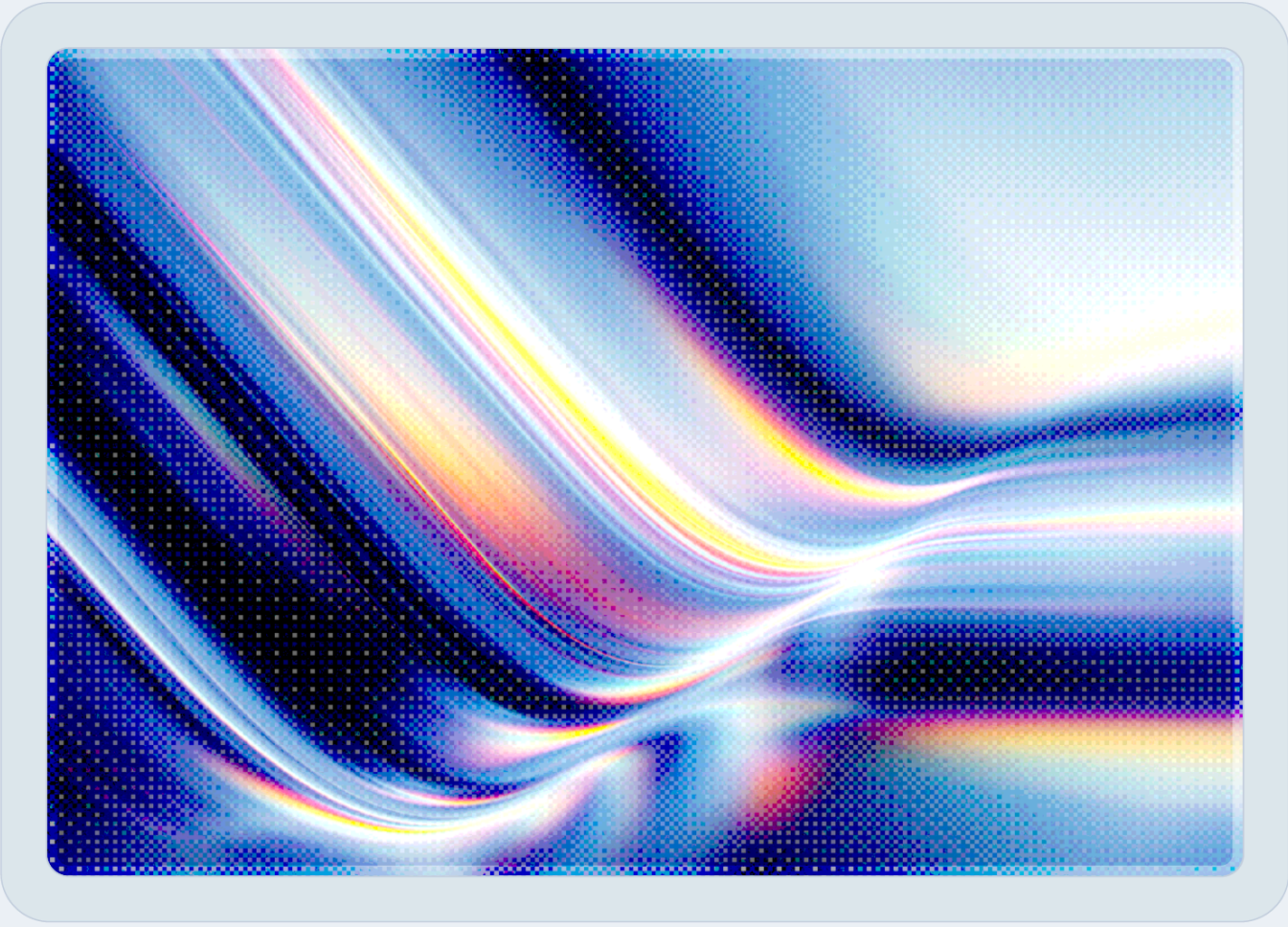
CSAI Foundation | Cloud Security Alliance

GitSpawn: How a Git Config File Hijacks AI Coding Agents

2026-09-03

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

Security researchers at Manifold Security disclosed a class of vulnerabilities, collectively named GitSpawn, in which a repository's own `.git/config` file can force a command-line AI coding agent to execute attacker-chosen code the moment the agent inspects the repository [1][2]. The flaw does not depend on a malicious dependency, a poisoned prompt, or a compromised MCP server; it exploits a legitimate Git performance setting, `core.fsmonitor`, that names a helper program Git runs automatically whenever an operation refreshes the repository index [2]. Because AI coding agents routinely run `git status` or `git diff` in the background to orient themselves inside a project, they trigger that helper program without the developer ever running a command directly.

Manifold reported eight related findings across seven widely used agents — Claude Code, OpenAI Codex, Cursor, Grok Build, Goose, Hermes Agent, and Qwen Code — and four remained unpatched at the time of publication [1][3]. The resulting code execution runs with the developer's full local privileges, outside whatever sandbox the agent otherwise enforces, and without any approval prompt appearing on screen [2]. On several affected tools the payload fires before the workspace-trust prompt is even accepted, meaning a developer who merely opens a suspicious project can be compromised before making any conscious decision to trust it [4]. This report explains the mechanism, summarizes patch status across the affected ecosystem, and connects the finding to CSA's existing guidance on sandboxing and configuration-file trust for agentic developer tools.

Background

Command-line AI coding agents such as Claude Code, Cursor, and OpenAI Codex have become widely used across many developers' workflows over the past two years, valued for their ability to read a codebase, run tests, and propose changes with minimal hand-holding. To do that job, an agent needs situational awareness: which branch is checked out, which files have changed, and what the repository's history looks like. Agents obtain that context largely the way a human developer would, by shelling out to Git itself, running commands like `git status`, `git diff`, or `git log` in the background before the developer has typed a single instruction [2].

Git's behavior on those commands is more configurable than a typical `.git/config` review would suggest. The `core.fsmonitor` setting, introduced as a performance optimization for large repositories, lets a project specify an external program that Git consults to determine which files have changed, instead of walking the whole working tree itself [2]. Critically, this setting lives in the repository's own `.git/config` file rather than in a global, developer-controlled location, and Git will invoke whatever command that file names during any operation that refreshes the index [3]. A repository does not need to be cloned through a trusted remote to carry this setting; the `.git` directory travels intact through a downloaded ZIP archive, a shared network drive, a USB stick, or any other channel that preserves the folder structure [2].

Manifold Security's research, published as the GitSpawn disclosure and covered by outlets including The Hacker News, GBHackers, and Cybersecurity News, found that this single configuration key gives an attacker a way to plant an arbitrary command that will run as soon as an AI coding agent opens the repository and performs its routine context-gathering [1][3][4]. Independent researchers filed overlapping findings with several of the same vendors, and OpenAI separately published three CVEs covering related Codex issues on the same day GitSpawn became public, indicating the underlying pattern was discovered along more than one path [1]. As of publication, no source has reported evidence of in-the-wild exploitation of any of the GitSpawn findings [1].

Security Analysis

The technical root cause is a trust boundary failure: AI coding agents treat a repository's Git configuration as benign operational metadata rather than as untrusted, attacker-influenced input, even though the configuration file travels with the repository and can be authored by anyone who produced the archive, drive, or repository the developer is about to open [2][3]. Any Git command that refreshes the index — which includes the routine `git status` and `git diff` calls agents issue for situational awareness — reads `core.fsmonitor` from that file and executes the named program, passing it the same privileges as the user running the agent [2]. Because this happens inside Git's own machinery rather than inside the agent's tool-execution layer, none of the confirmation dialogs, sandboxes, or permission prompts that agents apply to model-initiated shell commands ever engage; from the agent's perspective, it merely asked Git a question and Git answered it [2][4].

The practical consequence is that opening a malicious repository — not running any command inside it, not accepting any suggested code, simply having the agent look at the project — can be sufficient for code execution. On Claude Code and Hermes Agent, researchers found the payload fires before the workspace-trust prompt is accepted; on Qwen Code, before the user has even authenticated to the tool;

on Goose, before the tool ever contacts the model; and on Grok Build, on the very first keystroke inside the project, meaning the standard "review before you trust this folder" safeguard that vendors rely on as their primary defense does not apply to this class of attack [1]. An attacker who achieves this level of execution gains access to whatever the developer's own session can reach, including SSH keys, cloud credentials, shell history, and every other repository present on the same disk [2][4] — the same blast radius CSA has previously documented for other AI-coding-agent compromises that escape or bypass the intended sandbox boundary (see CSA Resource Alignment, below).

Patch status varies considerably across the affected ecosystem, and the table below reflects Manifold's disclosure timeline as of September 1, 2026.

Agent	Vulnerable setting / path	Disclosed	Status as of Sept. 1, 2026
Claude Code	`core.fsmonitor`	June 26, 2026	Patched (v2.1.196)
Claude Code	`ultrareview` command (separate config key)	July 15, 2026	**Unpatched** (v2.1.252)
OpenAI Codex	`core.fsmonitor`	July 20, 2026	Patched (CLI 0.131.0; Desktop 26.519.x)
Cursor	`core.fsmonitor`	July 8, 2026	Patched
Goose	`core.fsmonitor`	July 13, 2026	Patched (v1.44.0; CVE-2026-72718)
Hermes Agent	`core.fsmonitor`	July 20, 2026	**Unpatched** (v0.21.0; CVE-2026-71963)
Qwen Code	`core.fsmonitor`	July 7, 2026	**Unpatched** (v0.22.3)
Grok Build	`core.fsmonitor`	July 14, 2026	**Unpatched** (v1.0.13)

Version and CVE figures for Claude Code, Goose, Hermes Agent, Qwen Code, and Grok Build are drawn from Manifold's disclosure [2]; the OpenAI Codex build numbers (CLI 0.131.0, Desktop 26.519.x) are drawn from The Hacker News' reporting [1]; the overall disclosure timeline is corroborated across [1][2][3].

Two details are worth calling out beyond the raw patch matrix. First, the Claude Code case shows that patching the specific `core.fsmonitor` sink does not necessarily close the underlying vulnerability class: a second, distinct configuration-driven execution path tied to the `ultrareview` feature remained open more than six weeks after the first fix shipped, suggesting that in this instance the fix was scoped to the reported sink rather than to the broader assumption that repository-supplied configuration is safe to process [3][4]. Second, four of the eight disclosed findings were still unpatched at publication, meaning several of the most widely used AI coding agents — Claude Code alone accounts for more than 77 million npm downloads a month, and the affected tools collectively account for close to half a million GitHub stars — remain exposed to a technique that requires no social engineering beyond convincing a developer to open a folder [2].

Recommendations

Immediate Actions

Security teams should treat any repository obtained outside a direct, authenticated `git clone` from a known-good remote — including ZIP downloads, archives shared over email or chat, USB transfers, and mirrored copies — as untrusted input before allowing an AI coding agent to open it, since the `.git/config` payload requires the `.git` directory to arrive intact [2]. Where feasible, inspect `.git/config` for unexpected `core.fsmonitor` or other command-bearing settings before opening a suspect project in any agent, keeping in mind Manifold's finding that "any setting that names a program can run it" is not limited to `core.fsmonitor` alone [2]. Organizations running any of the four agents still unpatched as of this writing — Hermes Agent, Qwen Code, Grok Build, or Claude Code's `ultrareview` path — should confirm current version numbers against vendor advisories and restrict or disable the affected feature until a fix ships [1][3].

Short-Term Mitigations

Development teams should sanitize Git configuration state whenever an agent shells out for context, for example by invoking `git -c core.fsmonitor=false status` rather than trusting the repository's own settings, an approach Manifold specifically recommends to vendors [2]. Agent deployments should also run inside a sandbox or isolated execution environment that constrains what a Git-invoked helper process can reach, rather than relying solely on the agent's own tool-confirmation layer, since this vulnerability class demonstrates that Git-triggered execution bypasses that layer entirely

[3][4]. Where an organization's tooling supports it, disabling `core.fsmonitor` processing by default for any repository not explicitly marked trusted removes the specific sink Manifold documented without requiring a vendor patch.

Strategic Considerations

GitSpawn is best understood as one instance of a broader pattern documented across 2026: AI coding agents accumulate implicit trust in inputs — configuration files, tool descriptions, repository metadata — that were never designed to carry executable authority. The Claude Code case within this disclosure illustrates the risk directly: the vendor patched the specific `core.fsmonitor` sink while a second, architecturally identical path (`ultrareview`) remained open, suggesting that patches can be scoped to the reported sink rather than to the underlying trust assumption that produced it [3][4]. Organizations building governance around AI coding agents should treat repository configuration, not just repository content or dependencies, as a first-class attack surface subject to the same least-privilege and sandboxing requirements applied to any other untrusted input, and should require vendors to describe their configuration-handling posture as part of security reviews and procurement.

CSA Resource Alignment

GitSpawn's mechanism — a configuration file that triggers unsandboxed, unprompted code execution the moment an AI coding agent processes a repository — closely parallels findings CSA has already published on AI coding agent configuration trust and sandbox design, and this document's recommendations extend that existing body of work rather than introducing a new framework.

CSA Labs' "MCP Attack Surface: Tool Poisoning and IDE Auto-Execution" [6] documents the near-identical structural pattern of IDEs and agents auto-launching processes defined in a repository's committed configuration files with full developer privileges and no sandbox, describing an attacker who "could commit an innocuous initial configuration, wait for team members to approve it, and then replace the server command with an arbitrary payload." GitSpawn shows that this same failure mode extends beyond MCP configuration to Git's own configuration surface, reinforcing that the underlying gap is agents' broad trust in any repository-supplied configuration, not a defect specific to one protocol.

CSA Labs' "AI Coding Agent Sandbox Escapes: The Trust Handoff Flaw" [7] is directly relevant to why sandboxing alone did not stop GitSpawn: that research found that "sandboxes govern the agent's actions, not the downstream effects of the files the agent produces," and GitSpawn demonstrates a variant of that gap, in which Git — a trusted downstream process invoked by the agent — executes attacker-controlled configuration outside the agent's own sandbox boundary entirely. Organizations

applying that paper's recommendation to validate trust at every point where agent output or agent-invoked tooling touches the filesystem should extend that validation explicitly to Git's configuration-reading behavior.

CSA Labs' "GuardFall: Shell Injection Bypass Defeats AI Coding Agent Guardrails" [8] documents a related structural theme: guardrails built to inspect commands before execution can be defeated when the inspection point sits earlier in the pipeline than the actual point of execution. GuardFall's shell layer and GitSpawn's Git-configuration layer are different technical mechanisms, but both illustrate that agent vendors have repeatedly under-scoped which inputs their safety layer actually covers, leaving execution paths that bypass user-facing approval prompts entirely.

Finally, CSA's AI Controls Matrix (AICM) v1.1 provides the control vocabulary — spanning application and interface security, infrastructure and virtualization, and supply chain domains — that organizations can use to formalize configuration-file trust boundaries for AI coding agents as a standing control requirement rather than a one-off incident response [5].

References

[1] The Hacker News. "Malicious .git Configs Can Make Claude, Codex, Cursor, and Other AI Agents Run Attacker Code." The Hacker News, September 2026.

[2] Manifold Security. "GitSpawn: A Single Flaw Lets Untrusted Repos Run Code in Claude Code, Codex, Cursor, and Grok." Manifold Security Blog, September 2026.

[3] Cybersecurity News. "GitSpawn Flaws Let Malicious Repositories Execute Code in Claude Code, Codex, Cursor, and Grok." Cybersecurity News, September 2026.

[4] GBHackers. "GitSpawn Flaw Enables Arbitrary Code Execution in Claude Code, Codex, Cursor and Grok." GBHackers, September 2026.

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

[6] Cloud Security Alliance Labs. "MCP Attack Surface: Tool Poisoning and IDE Auto-Execution." CSA Labs, July 2026.

[7] Cloud Security Alliance Labs. "AI Coding Agent Sandbox Escapes: The Trust Handoff Flaw." CSA Labs, July 2026.

[8] Cloud Security Alliance Labs. "GuardFall: Shell Injection Bypass Defeats AI Coding Agent Guardrails." CSA Labs, July 2026.

