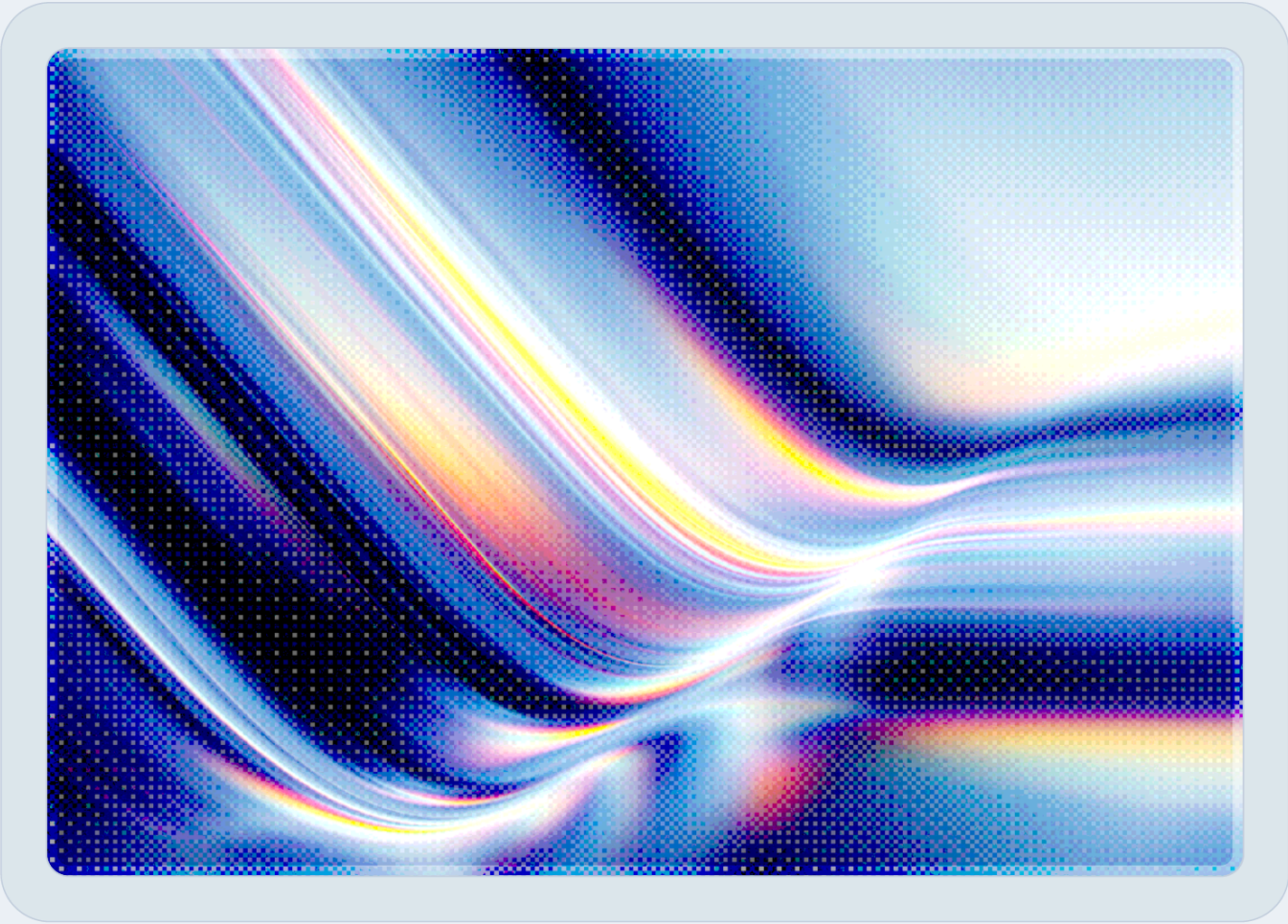
CSAI Foundation | Cloud Security Alliance

# Approved Once, Leaked Forever: Secrets in Claude Code Settings Files

How "Don't Ask Again" Permission Rules Turn Inline Credentials into Persistent, Publishable Artifacts

2026-11-08

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

**© 2026 Cloud Security Alliance. Some rights reserved.**

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

*This document was generated with AI assistance and has not undergone official CSA review and approval processes.*

## Key Takeaways

When a developer chooses "Yes, and don't ask again" on a Claude Code permission prompt for a shell command, Claude Code saves the approved command as an `allow` rule in the project's `.claude/settings.local.json` file [3]. If that command contained a password, token, or API key typed inline, the secret is saved in that file in plain text [2][5]. On November 6, 2026, Apiiro published research reporting that 1,122 of 104,185 public copies of this file on GitHub contained at least one secret. Those files held 1,379 distinct secrets, including 379 database passwords, 43 SSH logins for public servers, and 23 AWS keys [1][2].

The exposure is both cumulative and long-lived. Apiiro reported that every file it found with a secret was still on its repository's default branch, and in half of those repositories the file had been there for at least 217 days. Older versions in git history held another 734 secrets that no current copy contained [2]. Conventional secret scanners miss much of this material: gitleaks flagged 564 of the 1,122 files and TruffleHog flagged 643, but neither tool flagged 215 of them, mostly because the secrets were passwords embedded in command syntax such as `PGPASSWORD=`, `sshpass -p`, or `mysql -p` [2]. Git is also not the only way out. According to research by Check Point, as reported by SecurityBrief, 428 public npm packages shipped a `.claude/settings.local.json` file, and 33 of those files, across 30 packages, contained credentials [7].

These findings lead to two priorities for defenders. Rotation comes first, because deleting the file or rewriting history does not undo a leak; organizations should treat any credential that ever appeared in an agent settings file as compromised, rotate it, and only then clean up repositories and local files. Beyond remediation, organizations should treat agent configuration directories like `.env` files. Folders such as `.claude/` hold secrets and should be excluded from version control, package publication, container builds, and archives by default.

## Background

Claude Code, Anthropic's agentic coding tool, asks the user before it runs a shell command that has not already been approved. The permission prompt offers a one-time approval or a standing approval, labeled "Yes, and don't ask again." Claude Code's documentation says the standing approval is written as

an `allow` rule to `.claude/settings.local.json`, which the product calls "project local" settings, meaning personal overrides that apply to one user in one project [3]. In a git repository, Claude Code reads and writes the file at the repository root, so one approval applies across the whole repository [3]. Rules can match exact commands or patterns, and the permissions documentation explains how to write broader rules such as prefix matches [4]. Anthropic's documentation does not describe redaction of stored rules, and in practice, as reported in GitHub issue #5544 and reproduced by Apiiro, the rule saved by the "don't ask again" button records the full command text as it was approved [2][5]. Whatever the developer typed into that command, including a connection string with an embedded password or an `Authorization: Bearer` header, therefore becomes part of a persistent configuration file.

The documentation describes a safeguard that addresses one leak path. The first time Claude Code writes the file in a git repository that does not already ignore it, it adds `**/.claude/settings.local.json` to the user's *global* git excludes file, so git on that machine will not stage it [3]. The documentation also says that if a user creates the file by hand, the user must add it to `.gitignore` themselves [3]. As The Hacker News noted when reporting Apiiro's findings, the exclusion goes into the user's global git configuration and not into the project's `.gitignore` [2]. That distinction matters. A global exclude protects one developer on one workstation. It does not travel with the repository to a teammate's clone, a CI runner, a fresh laptop, a container image, or an npm tarball.

The problem has been reported publicly before. GitHub issue #5544 in the `anthropics/claude-code` repository, filed on August 11, 2025, reported that approved bash commands containing credentials were stored verbatim, in plain text, in `settings.local.json`. The reporter asked for secrets to be redacted or excluded from the allowlist [5]. An automated process marked the issue stale and closed it as "not planned" on January 3, 2026, and the page shows no maintainer response [5]. Issue #13106, filed on December 4, 2025, reported that a user's `settings.local.json` containing tokens had been committed and pushed to a public repository, which triggered a secret-scanning alert. It was closed as a duplicate of an issue reporting that Claude Code no longer added the file to `.gitignore` [6]. The Hacker News reported that the four related issues Apiiro cited were all closed by the same bot and that none of the reports received a reply from Anthropic [2].

# Security Analysis

## Scale and Composition of the Exposure

Apiiro searched public GitHub and found 104,185 copies of `.claude/settings.local.json` in 103,617 repositories. Of those, 1,122 copies, about 1.1%, contained at least one secret, and together they held 1,379 distinct secrets [1][2]. More than a third of the secrets gave direct access to a database; that category includes the 379 database passwords listed below along with other database credentials. The researchers also identified tokens for GitHub, GitLab, and npm, which would let an attacker reach source code and package publishing [1][2]. The owners of the affected files included 63 organizations, and one company that the researchers notified confirmed its exposed key had been live [2]. Apiiro did not test whether the remaining secrets still worked, so the number of currently exploitable credentials is unknown [2]. Even so, the presence of SSH logins for public servers, publishing tokens, and at least one confirmed-live key suggests that a meaningful share of this material is real infrastructure access, although some database passwords likely belong to local development instances.

These figures come from a single study by Apiiro, a commercial application-security vendor. CSA has not independently replicated the search, though the underlying storage behavior is documented in Anthropic's own issue tracker [5][6]. The table below summarizes the principal figures Apiiro reported, as published by The Hacker News.

| Measure | Reported value |
|---|---|
| Public copies of `.claude/settings.local.json` examined | 104,185 (in 103,617 repositories) |
| Copies containing at least one secret | 1,122 (about 1.1%) |
| Distinct secrets in current copies | 1,379 |
| Database passwords | 379 |
| SSH logins for public servers | 43 |
| GitHub, GitLab, and npm tokens | 42 |
| AWS keys | 23 |

| Measure | Reported value |
|---|---|
| Additional secrets found only in git history | 734, across 509 repositories |
| Forks carrying the same secrets | 211 forks of 74 repositories |
| Shared `.claude/settings.json` copies with secrets | 298 of 76,011 |
| Total, including history, shared settings, and agent chat logs | 2,539 secrets across 1,994 repositories |

*Source: Apiiro research as reported by The Hacker News [1][2].*

The final row combines current copies, git history, shared settings files, and agent chat logs committed to repositories; the published figures do not break out the chat-log share separately [2]. The shared project file, `.claude/settings.json`, is designed to be committed so a team gets the same permissions and hooks [3]. The 298 copies of that file containing secrets show that the problem is not limited to the local file. Developers sometimes promote approved commands into the shared file, and the documentation notes that in the VS Code extension, the approval card lets the user choose the shared project file as the destination for a rule [3]. In that case, a secret in an approved command goes straight into a file that is meant to be committed.

## Why Persistence Makes This Worse Than a Typical Leak

Several properties of this leak path make it more durable than the usual case of a key pasted into source code. The first is that the write happens as a side effect of a usability choice. A developer who clicks "don't ask again" is thinking about removing prompt friction, not about creating a credential store, and the documentation describes the file as a place for personal preferences [3]. The second is that the exposure persists after remediation attempts. Apiiro found 734 secrets in past versions that were no longer in any current copy, and 211 forks of 74 repositories carried the same secrets as the originals [2]. Removing the file from the default branch therefore does not remove the secret from history, and deleting a repository does not remove forks. The third is that the exposure is long-lived in practice. In half of the affected repositories, the file had been on the default branch for at least 217 days when Apiiro looked [2].

Apiiro's attribution data suggests that the agent itself is sometimes part of the commit path. In 539 of the 1,105 repositories the researchers traced, a commit that listed Claude as a co-author changed the file [2]. That figure does not establish that Claude Code introduced or pushed every exposed credential, and

the global-exclude safeguard should prevent the file from being staged on a correctly configured machine [3]. It does indicate that agent-driven commits appear regularly in the history of these files. One possible explanation, which CSA has not verified, is that agent-initiated commits may stage changes more broadly than a developer would manually. The researchers reproduced the storage behavior in current Claude Code releases running Opus 5.5 and Fable 5.1. Among 16 coding agents whose documentation they compared, Claude Code was the only one confirmed to save the full approved command in a project file by default [2].

## Leak Paths Beyond Git

Because the documented safeguard works at the git layer, any distribution channel that does not consult the user's global git excludes can carry the file out. The npm registry is the clearest documented example. According to research by Check Point, as reported by SecurityBrief, a scan of about 46,500 packages found 428 that contained `.claude/settings.local.json`, and 33 of those files, across 30 packages, contained credentials. The report lists the exposed material as including npm publishing tokens, GitHub personal access tokens, Telegram bot tokens, production bearer tokens, and Hugging Face API tokens [7]. npm decides what to publish from `.npmignore` or the `files` field in `package.json`. Neither excludes `.claude/` by default, and `npm publish` gives no warning when the directory is present [7]. A leaked npm publishing token is especially dangerous because, depending on the token type and the package's two-factor authentication settings, it can let an attacker push a malicious version of the package to its downstream users.

By the same logic, container builds that copy the build context without a `.dockerignore` entry, zip archives attached to support tickets, and backups or file-sync tools that ignore git rules may all carry the file. The npm research documents only the npm channel; the other channels are an inference from how those tools select files, not observed incidents. Organizations should check them anyway, because the precondition, a plaintext secret sitting in the working tree, is the same.

## Local Exposure and Credential-Harvesting Malware

A secret in `settings.local.json` is exposed even if the file is never committed. It sits in plaintext in the project directory, readable by any process running as the developer. The Hacker News noted that the Shai-Hulud worm of September 2025 ran TruffleHog on infected machines to harvest credentials, and that TruffleHog flags more than half of the affected files [2]. Credential-harvesting malware that reaches a developer workstation through a poisoned dependency or a malicious extension would therefore be likely to find these secrets. A related risk is that the coding agent can read its own configuration directory. Given the prompt-injection attacks against coding agents that CSA has

documented [8], it is plausible that injected instructions could direct an agent to read and exfiltrate this file. CSA has not identified a public report of that specific chain, so it should be treated as a credible scenario, not an observed attack.

### Detection Gaps

Standard secret scanners are tuned for high-entropy tokens with recognizable prefixes, such as AWS access keys or GitHub tokens. They are weaker at spotting low-entropy passwords inside command-line syntax. In Apiiro's sample, gitleaks and TruffleHog together still missed 215 of the 1,122 files. Most of the misses were passwords passed through `PGPASSWORD=`, `sshpass -p`, or `mysql -p` [2]. Because those are database and SSH credentials, the secrets scanners miss most often are also among the most directly useful to an attacker. Organizations that rely only on default scanner rulesets should assume they are under-counting their exposure in agent configuration files.

## Recommendations

The recommendations below are ordered by urgency. Immediate actions address credentials that may already be exposed, short-term mitigations prevent new exposures, and strategic considerations address how organizations select and govern AI coding tools.

### Immediate Actions

Security teams should start by inventorying exposure across their source control estate. They should search every organization-owned repository, including forks and archived repositories, for `.claude/settings.local.json` and `.claude/settings.json`, and scan full git history, not only the default branch [2]. They should also check published packages in npm and any internal registries for a `.claude/` directory, since a package published while Claude Code was in use may carry the file [7]. Every credential found in these files, whether in the current version or in history, should be rotated before any cleanup. Rewriting history or deleting the file leaves copies in forks, clones, caches, and registry mirrors [2]. Highest priority should go to credentials paired with a reachable public endpoint, such as a database host or SSH server named in the same command, and to package publishing tokens.

After rotation, developers should stop tracking the file with `git rm --cached .claude/settings.local.json`, add it to the repository's own `.gitignore` so the exclusion travels with the code [2], and confirm the rule with `git check-ignore -v`. They should also open

their local `settings.local.json` files and delete any `allow` rule that contains a literal secret, even if the file was never committed. The plaintext copy on disk is still a target for local malware [2].

## Short-Term Mitigations

To prevent new exposures, developers need to keep secrets out of approved commands. Commands should reference credentials through environment variables or a secrets manager, for example `$DB_URL` or a credential helper, instead of typing them inline. Developers should also prefer pattern-based permission rules such as `Bash(npm run *)` (equivalent to the older `Bash(npm run:*)` form) over approving single fully specified commands [2][4]. A pattern rule both reduces prompt fatigue and avoids recording the literal arguments of a specific invocation. Pattern rules carry their own cost, however, because they widen what the agent may run without confirmation. They should be scoped narrowly to low-risk command families, such as project build or test scripts; broad patterns on tools such as `curl`, `ssh`, or database clients trade a storage risk for an execution risk, especially where prompt injection is a concern.

Organizations should add `.claude/settings.local.json`, and ideally other agent state directories, to their standard `.gitignore`, `.npmignore`, and `.dockerignore` templates [2][7]. Packaging pipelines should include a pre-release check that lists package contents, for example with `npm pack --dry-run`, and fails the build if agent configuration files are present [7]. Secret scanning in pre-commit hooks and CI should cover full history and use custom rules for passwords inside command syntax, since default rulesets miss many of these cases [2]. Where Claude Code is centrally administered, managed settings can enforce deny rules that restrict the agent's file-reading tools from accessing `.env` files and secrets directories. Because those rules govern the file tools rather than every shell command, they should be paired with sandboxing or Bash restrictions for fuller coverage [3][4].

## Strategic Considerations

Although Apiiro confirmed this storage behavior by default only in Claude Code among the 16 agents it compared [2], the finding points to a class of risk that applies to any agentic tool that persists approval decisions: stored approvals can capture sensitive material. Organizations evaluating coding agents should ask vendors how standing approvals are stored, whether stored rules are redacted or abstracted into patterns, where the files live relative to the repository, and whether exclusion from version control is enforced at the project level or only on the user's machine. Apiiro's comparison suggests that design choices differ significantly across vendors [2], and these differences belong in procurement criteria.

Security teams should also treat agent working directories, such as `.claude/`, `.cursor/`, `.gemini/`, and `.specstory/`, as secret-bearing and include them in data-loss-prevention and scanning policies the same way they handle `.env` files [2].

Finally, the handling of public bug reports matters for enterprise risk management. The original report of plaintext storage was closed as stale by automation without a maintainer response [5], and The Hacker News reported that the related issues Apiiro cited were likewise closed by automation [2]. Organizations depending on AI coding tools should track the security issues filed against those tools and not assume a closed issue means a fixed one. They should also escalate through their vendor relationships when a known exposure path remains unaddressed.

## CSA Resource Alignment

This research note extends several recent CSA analyses of AI coding agents. CSA's *GhostApproval: A Trust Boundary Gap in Six AI Coding Agents* (July 2026) argued that human-in-the-loop approval is only as trustworthy as the information the approval interface presents and records [9]. The settings-file exposure is a companion failure on the other side of that boundary. GhostApproval showed approvals being redirected to actions the user did not intend. Here, the approval works as intended, but the record of it becomes a credential store. Together, the two findings suggest that the approval mechanism itself, including its UI, its persistence format, and its storage location, should be threat-modeled as a security-critical component of any coding agent.

CSA's *GhostCommit: Image-Based Prompt Injection Defeats AI Code Review* documents prompt injection that can direct coding agents to exfiltrate secrets [8]. That work underpins the local-exposure scenario described above: a plaintext credential in the agent's own configuration directory is precisely the kind of material an injected instruction could target.

CSA's *Sandboxing Agentic AI: Least-Privilege Patterns for MCP and Coding Agents* (May 2026) recommends per-task credentials over long-lived secrets and a project workspace that contains no long-lived secrets [10]. The settings-file leak shows what happens when that principle is not followed. A long-lived database password or publishing token typed into a command ends up persisted inside the workspace. The sandboxing note's identity-binding and capability-scoping guidance gives an architectural answer: if agents work with short-lived, narrowly scoped credentials brokered at runtime, then a recorded command has far less value to an attacker.

CSA's *AI Coding Agents: An Unaudited Supply Chain Node* (July 2026) described how stolen build and publishing credentials let attackers republish compromised packages [11]. The npm findings discussed here fit that pattern: settings files shipped inside public packages exposed npm and GitHub tokens that

could support exactly that kind of downstream compromise [7]. Finally, the *AI Controls Matrix (AICM) v1.1* gives the control framework for operationalizing these recommendations [12]. Its Identity and Access Management controls cover credential lifecycle and rotation, and its Threat and Vulnerability Management and Application and Interface Security controls cover secret scanning, secure build pipelines, and the evaluation of AI tooling in the development environment.

# References


[1] Apiiro. "Coding Agents Leaking Secrets to GitHub via 'Don't Ask Again'." Apiiro Blog, November 6, 2026.

[2] Khandelwal, Swati. "Claude Code's 'Don't Ask Again' File Exposes Passwords and Keys in 1,122 Public Copies on GitHub." The Hacker News, November 7, 2026.

[3] Anthropic. "Settings Files and Precedence." Claude Code Documentation, accessed November 8, 2026.

[4] Anthropic. "Configure Permissions." Claude Code Documentation, accessed November 8, 2026.

[5] asherfeldmangit. "[BUG] Approved Bash Commands Containing Credentials Are Stored As Is with Plain Text Credentials in settings.local.json (Issue #5544)." GitHub, August 11, 2025.

[6] andrehrferreira. "[BUG] Sensitive Tokens Stored in .claude/settings.local.json and Not Ignored by Git by Default (Issue #13106)." GitHub, December 4, 2025.

[7] SecurityBrief. "Claude Code Can Leak Secrets in Public npm Packages." SecurityBrief US, April 2026.

[8] Cloud Security Alliance. "GhostCommit: Image-Based Prompt Injection Defeats AI Code Review." CSA AI Safety Initiative, 2026.

[9] Cloud Security Alliance. "GhostApproval: A Trust Boundary Gap in Six AI Coding Agents." CSA AI Safety Initiative, July 15, 2026.

[10] Cloud Security Alliance. "Sandboxing Agentic AI: Least-Privilege Patterns for MCP and Coding Agents." CSA AI Safety Initiative, May 30, 2026.

[11] Cloud Security Alliance. "AI Coding Agents: An Unaudited Supply Chain Node." CSA AI Safety Initiative, July 8, 2026.

[12] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, accessed November 8, 2026.