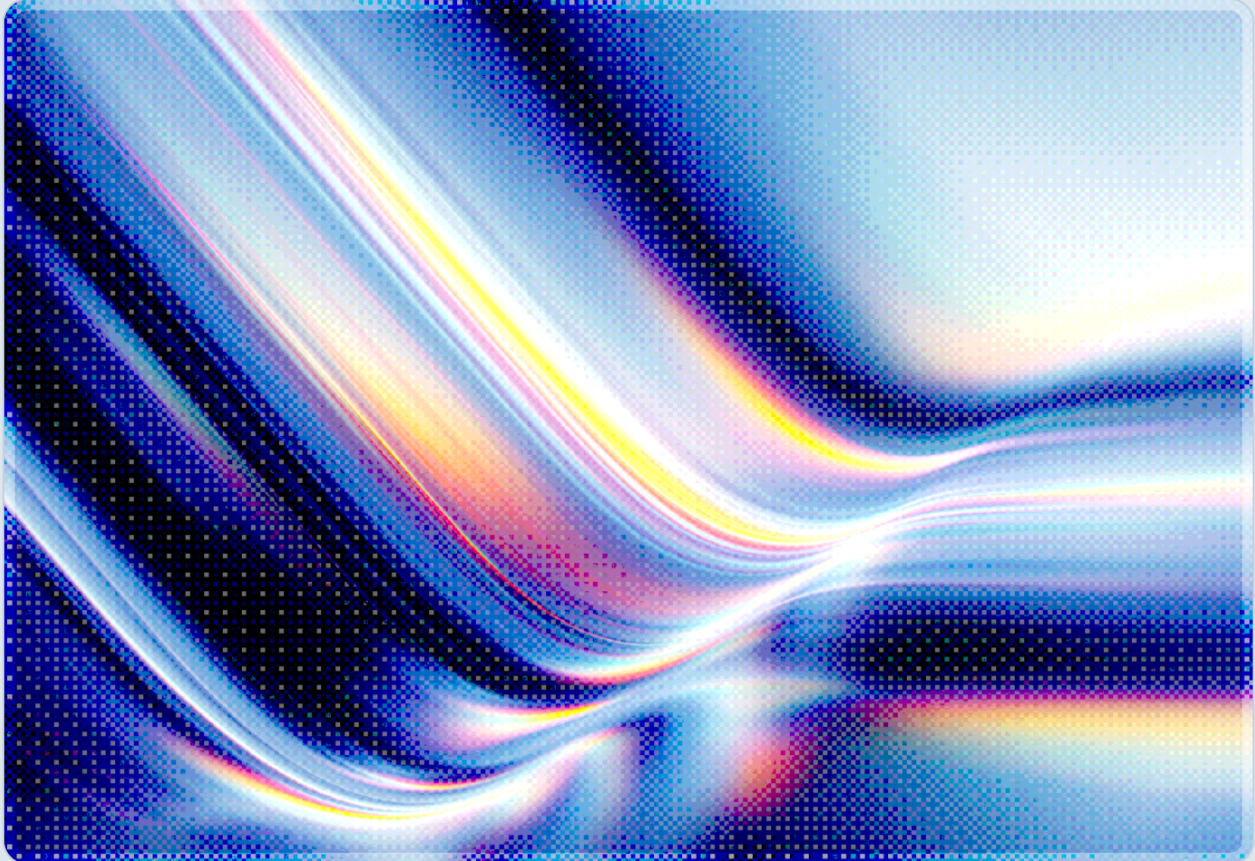


The Unrevoked Key: Agent Credential to LiteLLM Release Hijack

How a leaked agent platform master key became the LiteLLM v1.107.0 supply chain compromise

2026-10-24

 AI-assisted Rapid Research



CSAI

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

On October 21, 2026, a malicious `litellm==1.107.0` was published to PyPI and to LiteLLM's official Docker Hub and GHCR images. The PyPI wheels were live from 11:39 to roughly 13:23 UTC, and the affected Docker tags served the poisoned build until 14:20 UTC [1]. The payload installs a `.pth` hook that runs a credential stealer every time Python starts, even if LiteLLM is never imported. It also dumps upstream provider keys from the LiteLLM proxy database and starts a detached loader that polls hourly for remote code [3][4]. Because LiteLLM is a widely used AI gateway, with Datadog citing about 23 million weekly downloads [4], LiteLLM advises that any organization that pulled an unpinned install or a floating Docker tag during that window should assume its credentials were stolen [1].

LiteLLM's attack-chain disclosure traces the compromise to a single credential. An Agent Platform master key was committed to a public repository on May 27, 2026, and was never revoked. The commit was force-pushed over, but the key stayed valid for about four and a half months [2]. The attacker used that key on October 10 to query the internal agent platform's API, which returned a GitHub CLI OAuth token belonging to a GitHub organization owner. That token became the foothold for collecting more maintainer tokens over the following eleven days [2].

The attacker did not bypass LiteLLM's hardened release pipeline. They drove it. With stolen owner and maintainer tokens, they disabled branch protection, changed the release workflow so it would accept a fork as its source, and pointed it at a lookalike repository, `Berri-AI/litellm`. The legitimate pipeline then built, signed, and published the artifacts with valid provenance [2][3]. Days earlier, LiteLLM had relaxed its production gate, which normally required an approval account bound to an office device, so that any two engineers could release; LiteLLM notes that the stolen owner token could have removed that restriction in any case [2]. The controls LiteLLM had added after its March 2026 compromise (a private release repository, PyPI Trusted Publishing, immutable tags, and image signing) did not prevent this, because each of them ultimately deferred to holders of sufficiently privileged tokens [2][3].

For security teams, the lessons are about credentials, not about the package. Agent platforms that store or relay human developers' OAuth tokens are high-value identity stores. A secret removed from git history without being revoked is still exposed. And release pipelines that grant authority to any holder of two developer tokens have no defense once those tokens leak. Organizations should rotate everything an affected host could reach, inventory the credentials their agent tooling holds, and require release approvals that stolen tokens alone cannot satisfy.

Background

LiteLLM, maintained by BerriAI, is an open-source Python SDK and proxy that puts a single API in front of many LLM providers. When deployed as a proxy, it typically stores provider API keys, virtual keys, a master key, and a database connection string. That makes it a credential broker for every model provider an organization uses. This is the second LiteLLM supply chain compromise of 2026 [7]. On March 24, attackers associated with the TeamPCP campaign published malicious versions 1.82.7 and 1.82.8 to PyPI. They used a publish token reportedly harvested when LiteLLM's CI pipeline ran a backdoored Trivy scanner [9]. LiteLLM then moved releases to a private repository and adopted PyPI

Trusted Publishing, immutable Docker tags, and image signing [1][3]. Through the spring, the LiteLLM proxy was also the subject of a CISA KEV-listed command injection flaw and several chained authentication bypasses, which CSA analyzed in June [8]. The Hacker News reports that Socket found no evidence linking the October incident to TeamPCP [6].

The October incident unfolded in two phases, about eleven days apart. In the first phase, the attacker turned one leaked credential into several. In the second, they used those credentials to drive the release. LiteLLM's account of the root cause also changed as its investigation went on. Its first incident post suggested the GitHub tokens came from the logs of an internet-exposed internal LiteLLM deployment. In that post, LiteLLM reported observing "specific situations in which Claude Code will embed a user's GitHub CLI OAuth token into a response body," so that gateways which log response bodies could capture those tokens [1]. The post was updated on October 23 to note that parts of its initial account were inaccurate or incomplete [1]. The later attack-chain disclosure is more specific. It names the leaked Agent Platform master key as initial access and says the agent platform's own API returned the owner's GitHub token [2]. Both accounts point to the same structural problem: AI agent infrastructure ended up holding developers' high-privilege GitHub credentials in places an attacker could reach. This note follows the attack-chain disclosure, which LiteLLM describes as the canonical record. LiteLLM states that its investigation is complete, and that earlier updates written mid-investigation contained assumptions later found to be inaccurate or incomplete [2].

The table below summarizes the key events from LiteLLM's published timelines.

Date / Time (UTC)	Event
May 27, 2026, 18:53	Agent Platform master key committed in <code>opencode.json</code> to the public <code>LiteLLM-Labs/lite-harness</code> repository; the commit is later force-pushed over, but the key is not revoked [2]
Oct 10, 21:13–21:28	Attacker calls the Agent Platform API with the master key; <code>GET /api/v1/managed_agents/agents</code> returns an org owner's GitHub CLI OAuth token [2]
Oct 10, 22:11–22:14	Temporary-branch workflows in <code>BerriAI/litellm</code> and <code>BerriAI/litellm-auto</code> post repository secrets, including two more maintainers' tokens, to <code>webhook.site</code> ; the runs are then deleted [2]
Oct 12, 12:08–12:30	Second round of secret exfiltration across private repositories [2]
Oct 19	Release approval rule relaxed from an engineer plus an office-device-bound approval account to "any two engineers," to allow backports [2][6]
Oct 21, 10:08	Branch protection rejects a workflow change; the owner token disables protection and force-pushes it [1][2]
Oct 21, 10:46–11:11	Staging run built from <code>Berri-AI/litellm</code> at commit <code>20535d8f</code> ; the production gate is approved with a second maintainer's token [1][2]

Date / Time (UTC)	Event
Oct 21, 11:39–11:40	Seven malicious wheels published to PyPI [1]
Oct 21, 11:46	Docker Hub and GHCR image tags repointed to the malicious build [1]
Oct 21, 11:53–11:54	Workflow run records deleted using a third maintainer's token [1][2]
Oct 21, ~12:06	A community member opens a GitHub issue identifying the compromise [2]
Oct 21, ~13:23 / ~13:51 / 14:20	PyPI quarantines the release; GHCR tags unavailable; all affected Docker Hub tags removed or repointed [1]

Security Analysis

A Leak That Was Cleaned Up but Not Revoked

The root failure is a common one. A secret was exposed in a public repository, someone noticed, and the response was to rewrite git history instead of revoking the credential. LiteLLM states plainly that "key revocation did not occur for the master key" [2]. Force-pushing over a commit does not remove the secret from forks, clones, caches, or the many automated scrapers that watch public GitHub pushes. Once a secret has been public, it should be treated as compromised no matter what happens to the repository afterward. About four and a half months passed between the leak and its first observed use. That gap suggests the key may have been harvested early and held, although the disclosure does not say when the attacker obtained it. LiteLLM describes the service as an "internal alpha" [2]. The missed revocation suggests that its master key was not covered by an inventory or revocation-tracking process that would have caught the gap.

Agent Platforms as Credential Aggregators

The second failure is architectural, and its implications extend beyond LiteLLM. The Agent Platform is described as an open-source tool for orchestrating agents. It held, and returned through a list endpoint, a GitHub CLI OAuth token belonging to a human organization owner [2]. Agent orchestration systems that act on a developer's behalf create pressure to store user tokens alongside agent configurations. When that happens, one platform credential can unlock many human credentials, and the platform's list and read endpoints become a way to harvest them. LiteLLM's earlier hypothesis, that gateway logs captured tokens embedded in coding-agent response bodies, is no longer its account of

what happened, but it describes a plausible related risk: agent tooling can move human secrets through infrastructure that was never designed to protect them [1][2]. Either way, the agent platform's identity boundary was weaker than the GitHub organization's, and the attacker crossed from one to the other.

GitHub CLI OAuth tokens made the escalation worse. They are long-lived and broadly scoped, and LiteLLM notes that they are not subject to some of GitHub's re-authentication prompts for sensitive actions [2]. The owner's token was not scoped down or tied to a service account. As a result, it could disable branch protection, force-push to `main`, and dispatch workflows without any step-up challenge [2]. With that token, the attacker also ran temporary-branch workflows that printed repository secrets to an external endpoint and then deleted the run records. That is how one stolen token became three [2].

Driving the Pipeline Instead of Bypassing It

The release phase shows why provenance and signing do not prove intent. The attacker committed a change titled "ci: release from a fork source repository." That change let the workflow build from an arbitrary source repository, which they set to their `Berri-AI` fork [2]. LiteLLM notes that the malicious commit "exists only on the attacker's fork" but appears under the `BerriAI/litellm` URL, because GitHub serves fork commits through the parent repository's network [2]. The production gate normally requires an authorized engineer plus a dedicated approval account tied to an office device. Earlier that week it had been relaxed so that "any two engineers" could trigger and approve a release, and the attacker used two stolen tokens to meet that rule [2]. LiteLLM believes the attacker used separate tokens on purpose so the audit log would look like routine activity [2].

The results were artifacts that, according to StepSecurity, were "built, signed, and published by the legitimate pipeline with valid provenance" [3]. The source tag on GitHub stayed clean [1][5]. The SLSA provenance on the Docker images did record the hyphenated source repository, and investigators used it to trace the release [3]. Provenance was accurate, but nothing verified it at publish time against an expected source repository. StepSecurity also notes that the "Verified" badge on the malicious commits showed only that GitHub's web editor created them, not who was holding the token [3].

Payload and Blast Radius

The table below summarizes the payload components reported by independent analysts.

Component	Behavior	Key Indicators
<code>cook.pth</code> / <code>cook.py</code>	Runs on every Python interpreter start; a 61 KB stealer with 16 collection modules covering cloud, SSH/GPG, browser, CI, Kubernetes, and AI coding tool credentials, plus every readable <code>/proc/<pid>/environ</code> [3][4]	<code>PRIVCOLLECT_ACTIVE=1</code> , <code>/tmp/.pc_once_*</code>
Proxy database dump	If <code>DATABASE_URL</code> is set, reads <code>LiteLLM_CredentialsTable</code> and	Fires on first start of <code>litellm-database</code> image [3]

Component	Behavior	Key Indicators
	<code>LiteLLM_ProxyModelTable</code> to extract upstream provider keys [3]	
Exfiltration	AES-256-CBC bundle with RSA-4096-wrapped key, POSTed to <code>litellm[.]dev/healthcheck</code> [3][4]	<code>litellm.dev</code> , <code>www.litellm.dev</code> , <code>X-Token</code> header
<code>sitecustomize.py</code> loader	Forks a detached daemon that hourly reads a host from a GitHub dead-drop and <code>exec()</code> s <code>http://<host>/codepy</code> [3]	<code>/tmp/remote_code_loader.lock</code> , requests to <code>/codepy</code>

Datadog reports that the dead-drop file contained only test values and that no second-stage payload was observed [4]. StepSecurity warns, however, that anyone who later registers the deleted GitHub account could reactivate surviving loaders, so killing the daemon is required [3]. The stealer's encryption means packet captures cannot show what was taken [3]. According to The Hacker News, Wiz observed attackers validating stolen AWS instance role credentials from compromised CI runners and Kubernetes nodes about four and a half hours after exfiltration [6]. The likely blast radius therefore extends from developer laptops into CI and production clusters, and through the proxy database to every model provider account a LiteLLM deployment brokers.

Recommendations

Immediate Actions

Organizations should first find out whether they were exposed. Search lockfiles, pip caches, container registries, and deployment manifests for `litellm==1.107.0` or any pull of the `v1.107.0` , `latest` , `main-latest` , or `main-stable` tags between 11:46 and 14:20 UTC on October 21 (Docker Hub; GHCR tags were unavailable by about 13:51) [1]. On affected hosts, kill the process holding `/tmp/remote_code_loader.lock` before running pip or any other Python command, because every interpreter start re-runs the payload [1][3]. Then remove `cook.pth` , `cook.py` , `sitecustomize.py` , and their compiled `.pyc` files. It is preferable to rebuild environments and images from scratch and pin `litellm==1.106.2` or an earlier digest-pinned image [1].

LiteLLM advises that the absence of marker files does not prove a host is clean, and that any affected host with outbound internet access should be treated as having exfiltrated successfully [1]. From a clean machine, rotate instance and task IAM roles, every provider key stored in the LiteLLM proxy database, LiteLLM master and salt keys, the `DATABASE_URL` password, and any secrets present in process environments [1][3]. Block `litellm[.]dev` at egress and hunt for requests to `/codepy` [3][4]. Review CloudTrail and equivalent logs for `GetCallerIdentity` calls and other use of rotated credentials from outside your infrastructure [1].

Short-Term Mitigations

Every organization running agent orchestration platforms, AI gateways, or coding-agent infrastructure should apply the lessons of the root cause to itself. Inventory where those systems store or log human OAuth tokens, especially GitHub CLI tokens. Then either eliminate that storage or move it to short-lived, narrowly scoped tokens issued to dedicated service identities. Gateways that log request or response bodies should redact credential patterns before writing logs, and should restrict log access as tightly as the secrets store. Any secret discovered in a public repository should be revoked first and removed from history second. Secret-scanning alerts should be closed only once revocation is confirmed, not once the commit is gone.

Organizations that publish packages should limit release workflows to an allowlist of source repositories and fail closed if the source is a fork. They should also verify provenance against the expected repository before upload, not only after the fact. Release approvals should require a second factor that a stolen token cannot satisfy, such as hardware-bound approval accounts, which LiteLLM had in place before relaxing them [2]. Temporary relaxations of release controls should be time-boxed, logged, and watched for unusual approvers. Finally, alert on branch-protection changes, force-pushes to protected branches, and deletion of workflow runs. Each of these occurred during this incident [2], and each is uncommon enough in normal operations to serve as a useful detection signal.

Strategic Considerations

The incident shows that AI agent infrastructure is now part of the software supply chain's identity plane. Agent platforms, gateways, and coding assistants routinely act with delegated developer authority, so their credentials should be governed as non-human identities: each should have a named owner, an expiry, an inventory record, and a tested revocation procedure. Consumers of open-source AI tooling should stop relying on floating tags and immediate upgrades. Pinning by digest, applying a short cooldown before adopting new releases, and using malware scanning on intake all help; Chainguard reports its scanner flagged this release about 16 minutes after the first wheel appeared [5]. Software producers should treat signing and provenance as evidence to be checked against policy, not as proof of legitimacy. A pipeline that faithfully signs whatever authorized tokens tell it to build protects consumers only as well as those tokens are protected.

CSA Resource Alignment

This incident follows directly from CSA's earlier LiteLLM analyses. [LiteLLM PyPI Backdoor: Credential Theft in AI Toolchains](#) [9] documented the March TeamPCP compromise and the same `.pth`-based persistence technique. [LiteLLM AI Gateway: KEV-Listed Attack Chain Enables Full Takeover](#) [8] and [LiteLLM AI Gateway: Critical Vulnerability Chain Exposes API Keys](#) [13] framed the AI gateway as a credential broker whose compromise exposes every connected provider key. The October payload's targeted dump of the proxy credentials table is consistent with that framing. The October incident also shows a different weak point: the earlier notes centered on stolen publish tokens and runtime vulnerabilities, while this one exploited the identity layer behind a hardened pipeline.

[AI Coding Agents: An Unaudited Supply Chain Node](#) [10] argued that AI agents and their integrations have become supply chain dependencies that organizations are not inventorying or auditing. LiteLLM's experience is a concrete case. An agent orchestration platform held an organization owner's GitHub token, and that agent infrastructure became the path

into the release process. Agent tooling can also move such tokens through gateway logs, a related risk class that organizations should account for in the same inventory. [The Non-Human Identity Governance Vacuum](#) [11] recommends an NHI registry, zero standing privilege for agents, and credential rotation with rapid revocation. An unrevoked master key for an alpha service, and a broadly scoped human token stored inside an agent platform, are the kinds of gaps those controls target.

At the control level, CSA's [AI Controls Matrix \(AICM\) v1.1](#) [12] covers the relevant requirements across its Identity and Access Management domain (credential lifecycle, least privilege, revocation), its Supply Chain Management, Transparency, and Accountability domain (provenance and dependency integrity), and its Application and Interface Security domain (secure build and release). Organizations can use AICM to check whether their agent platforms and release pipelines meet these requirements, and STAR for AI assessments to ask the same of AI tooling vendors.

References

- [1] LiteLLM. "[Contained: Oct 21, 2026 LiteLLM release v1.107.0 Compromised.](#)" LiteLLM Blog, October 21, 2026.
- [2] LiteLLM. "[The attack chain behind the v1.107.0 LiteLLM release.](#)" LiteLLM Blog, October 23, 2026.
- [3] StepSecurity. "[litellm 1.107.0 Compromised on PyPI: Backdoored Wheels Drop a Credential Stealer and an Hourly Remote Code Loader.](#)" StepSecurity Blog, October 21, 2026.
- [4] Datadog Security Labs. "[LiteLLM supply chain compromise: What you need to know.](#)" Datadog, October 21, 2026.
- [5] Chainguard. "[LiteLLM attacked again: Malicious 1.107.0 wheels on PyPI and Docker Hub \(October 2026\).](#)" Chainguard Unchained, October 21, 2026.
- [6] The Hacker News. "[LiteLLM Suspects GitHub Token Leak in AI Gateway Logs Behind Malicious Package Release.](#)" The Hacker News, October 22, 2026.
- [7] Help Net Security. "[LiteLLM hit by second supply chain attack this year, users urged to rotate credentials.](#)" Help Net Security, October 22, 2026.
- [8] Cloud Security Alliance. "[LiteLLM AI Gateway: KEV-Listed Attack Chain Enables Full Takeover.](#)" CSA AI Safety Initiative, June 17, 2026.
- [9] Cloud Security Alliance. "[LiteLLM PyPI Backdoor: Credential Theft in AI Toolchains.](#)" CSA AI Safety Initiative, March 27, 2026.
- [10] Cloud Security Alliance. "[AI Coding Agents: An Unaudited Supply Chain Node.](#)" CSA AI Safety Initiative, July 8, 2026.
- [11] Cloud Security Alliance. "[The Non-Human Identity Governance Vacuum.](#)" CSA AI Safety Initiative, May 20, 2026.
- [12] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1.](#)" Cloud Security Alliance, June 22, 2026.
- [13] Cloud Security Alliance. "[LiteLLM AI Gateway: Critical Vulnerability Chain Exposes API Keys.](#)" CSA AI Safety Initiative, June 16, 2026.