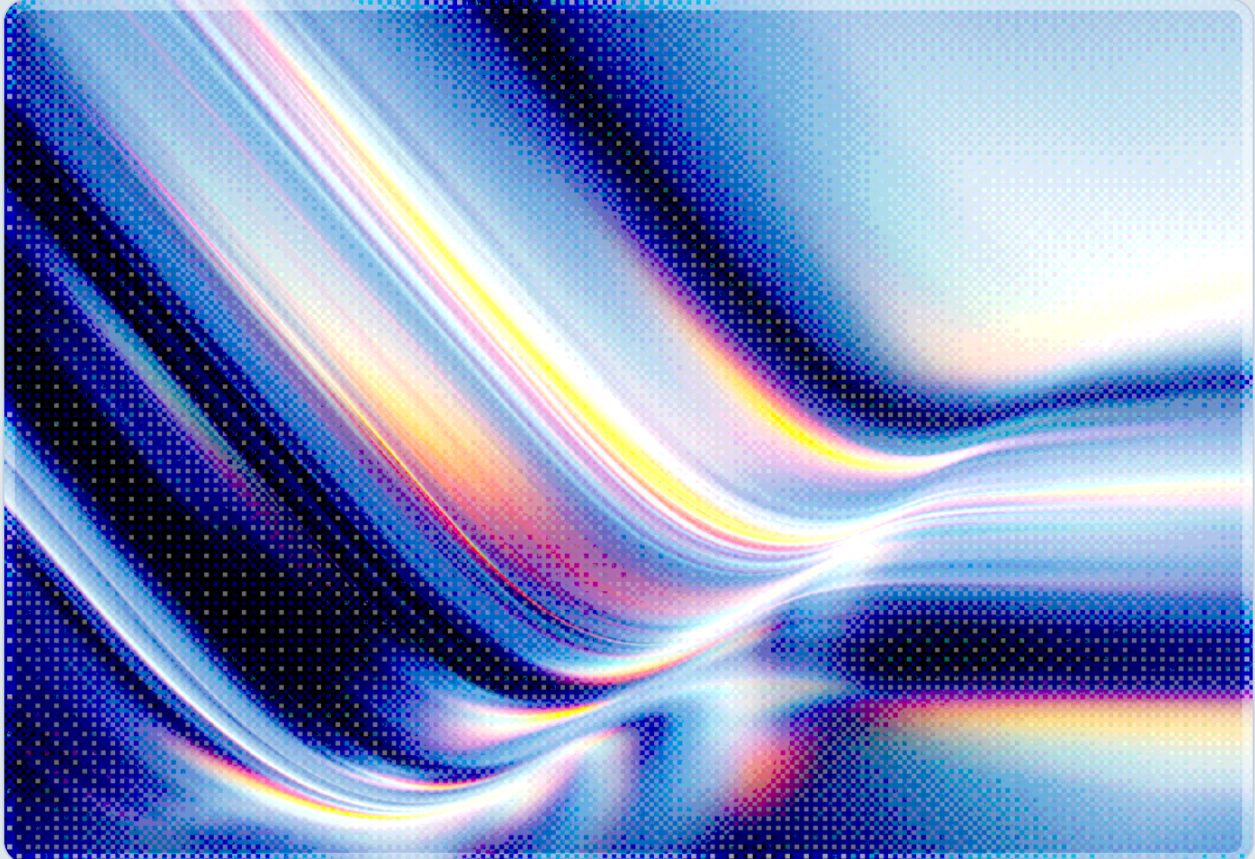


LiteLLM 1.107.0: An Unrevoked Agent Key Becomes a Release Takeover

Security Implications and Guidance for Credential Revocation,
Agent Platforms, and Release Pipelines

2026-10-26

 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, an attacker published a malicious `litellm==1.107.0` to PyPI and pointed LiteLLM's official Docker Hub and GitHub Container Registry (GHCR) tags at a poisoned image. The seven PyPI wheels were available from 11:39:51 UTC until PyPI removed them at 13:23 UTC. The affected Docker Hub tags (`v1.107.0`, `latest`, `main-latest`, and `main-stable`) served the malicious image until 14:20 UTC [1]. The payload runs whenever Python starts, whether or not LiteLLM is imported. It collects cloud, developer, AI-tool, browser, and LiteLLM proxy database credentials, and on Unix hosts it starts a detached loader that polls hourly for remote code [3][4]. LiteLLM, StepSecurity, and Microsoft agree that any host which installed the release should be treated as compromised. All three say that removing the package does not clean the host [1][3][4].

LiteLLM's root-cause disclosure traces the incident to one missed step. On May 27, 2026, the master key for an experimental LiteLLM Agent Platform deployment was committed to a public repository. The team force-pushed over the commit, but in LiteLLM's words, "key revocation did not occur for the master key" [2]. Four and a half months later, the attacker used that key to call the Agent Platform API, which returned an unscoped GitHub CLI OAuth token belonging to a GitHub organization owner [2]. That owner-level access was enough to collect more maintainer tokens, override branch protection, and drive LiteLLM's own release pipeline from a lookalike repository [2][6].

This note focuses on what security teams should take from that chain. A secret that has been exposed stays exposed until it is revoked, however the git history is cleaned up afterward. Agent platforms that hold human developers' tokens become credential aggregators. Release controls that any two stolen developer tokens can satisfy do not hold up once those tokens leak. CSA recommends that organizations treat revocation as a tracked, verifiable control rather than a cleanup step. They should also inventory the tokens 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 interface in front of many large language model providers. In proxy deployments, it typically stores upstream provider API keys, virtual keys, a master key, and a database connection string [17]. That makes a LiteLLM host a high-value target, because one compromised gateway can expose an organization's access to every model provider it uses. CSA has examined this concentration of credentials several times in 2026, including the KEV-listed command injection chain in June [15] and the persistence of default admin keys on exposed gateways in September [17].

This is the second LiteLLM supply chain compromise of 2026. In March, the TeamPCP campaign published malicious versions 1.82.7 and 1.82.8 after stealing CI/CD credentials through a backdoored Trivy scanner. CSA's analysis of that campaign notes that it began when the attackers exploited an *incomplete credential rotation* at Aqua Security [14]. After the March incident, LiteLLM moved releases to a private repository, adopted PyPI Trusted Publishing, made Docker release tags immutable, and began signing images [1]. JFrog reports that no evidence currently links the October incident to TeamPCP and advises treating the two as separate campaigns [7]. They have something important in common, though, even though the failures occurred at different organizations. In both, a credential that its owners knew had been exposed but did not fully invalidate gave the attacker its opening.

LiteLLM's explanation changed as its investigation progressed. Its first incident post suggested that the attacker had obtained GitHub tokens from the logs of an internet-exposed internal LiteLLM deployment. The post was later updated to say that parts of that account were inaccurate or incomplete [1]. The attack-chain disclosure published on October 23, which LiteLLM treats as the canonical record, identifies the leaked Agent Platform master key as the initial access vector. According to that disclosure, request logs show the first token came from the Agent Platform API, not from gateway logs [2]. This note follows the October 23 account. The table below combines the timelines LiteLLM published.

Date / Time (UTC)	Event
May 27, 2026	Agent Platform master key committed to the public <code>LiteLLM-Labs/lite-harness</code> repository; the commit is force-pushed over, but the key is not revoked [2][6]
Oct 10, 21:27	Attacker calls <code>GET /api/v1/managed_agents/agents</code> with the master key; the response contains an organization owner's GitHub CLI OAuth token [2]
Oct 10	Throwaway <code>ci-tmp-*</code> branches in <code>BerriAI/litellm</code> and <code>BerriAI/litellm-auto</code> run workflows that post repository secrets, including two maintainers' tokens, to <code>webhook.site</code> ; the runs are deleted [2][6]
Oct 12	A second round of workflows exfiltrates all repository secrets from several private repositories [2]
Oct 21, 10:08	Branch protection rejects a release-workflow change; the owner token is used to override protection and apply it [1][2]
Oct 21, 10:46–11:11	Staging run builds from lookalike <code>Berri-AI/litellm</code> ; a second maintainer's token satisfies the production approval gate [1][2]
Oct 21, 11:39–11:46	Seven malicious wheels published to PyPI; container publication jobs push the malicious image [1]
Oct 21, 11:53–11:54	A third credential is used to delete production and staging workflow run records [1]
Oct 21, ~12:06	Compromise reported publicly by an outside researcher [6][8]
Oct 21, 13:23–14:20	PyPI removes the release; GHCR tags unavailable by 13:51; all Docker Hub tags removed or repointed by 14:20 [1]

Security Analysis

Revocation Is the Control, Not the Cleanup

The first failure in the chain follows a pattern widely reported in secret-exposure incidents, and this incident shows how much it can cost. When the master key was committed, LiteLLM caught the mistake and rewrote the repository history. The Hacker News reports that the exposed keys were flagged for revocation, but the master key itself was never revoked [6]. GitHub's documentation on removing sensitive data addresses this case directly. It says that when the data is a credential, "as a first step you need to revoke and/or rotate that secret," and that once the secret is revoked, rewriting history "may not be warranted" [9]. A force-push does not remove a secret from existing clones, forks, caches, or the automated scrapers that watch public commits [9]. It only removes the secret from the place the defender looks.

The gap between the May 27 exposure and the first observed use on October 10 is about four and a half months. The disclosure does not say when the attacker obtained the key, so it is unclear whether the key was harvested early and held or found later [2]. In either case, the key stayed valid long enough for both to be possible. The likely process failure is that the revocation was flagged but never verified as

done. Revocation tasks that live in a ticket or someone's memory can easily drop out once the visible problem, the public commit, has gone away. TeamPCP's March campaign began the same way, with a rotation that was started but not finished [14]. Organizations should treat both incidents as evidence that revocation must be tracked through to confirmed completion, and must be easy to check from outside the team that requested it.

Agent Platforms as Credential Aggregators

The second failure is architectural, and it reaches well beyond LiteLLM. The Agent Platform was an experimental system for running self-hosted cloud agents that open pull requests. To do that, it was configured with a GitHub CLI OAuth token belonging to an organization owner. LiteLLM states that this token "was not associated with a service account and was not scoped down" [2]. The platform also returned the token in an ordinary list-agents API response. As a result, anyone holding the platform's master key could collect a human owner's GitHub credential with a single request [2].

This pattern will likely recur as organizations deploy coding and operations agents. An agent that acts for a developer needs credentials, and the easiest credential to hand it is the developer's own token. The platform then holds human-grade privilege behind a separate, often weaker authentication boundary: here, a static master key for an alpha service. LiteLLM also notes that GitHub CLI tokens are issued after two-factor authentication. They are not subject to GitHub's re-authentication prompts for sensitive actions, so a stolen token carries the owner's full authority without further challenge [2]. LiteLLM's earlier hypothesis has been superseded, but it pointed to a related risk. When agent tooling passes human secrets through gateways and logs, those systems become credential stores that were never designed to protect them [1].

Driving the Pipeline Instead of Bypassing It

LiteLLM's post-March hardening was meant to make unauthorized publication harder, and in a narrow sense it worked. The attacker did not steal a PyPI token or forge a signature. Instead, they used legitimate identities to make the legitimate pipeline do the work. With the owner token, the attacker overrode the branch protection that had rejected the workflow change, then redirected the release source to `Berri-AI/liteLLM`, which differs from the real organization name by one hyphen [1][2]. LiteLLM's normal process required an authorized engineer to trigger a release and a special approval account tied to a physical device in the office to approve it. Earlier that week, the rule had been relaxed to "any two engineers" to support more frequent releases and backports [2]. The attacker used one stolen maintainer token to trigger the release and another to approve it. LiteLLM notes that the owner token could have removed the restriction outright, and suggests the attacker used separate identities to make the audit trail look less unusual [2].

As a result, artifacts built from attacker-controlled source passed through LiteLLM's Trusted Publishing and signing steps [1][4]. Provenance controls such as SLSA and PyPI Trusted Publishing show which pipeline produced an artifact [11][12]. They do not show that the pipeline's configuration, source repository, and approvers were the ones the project intended. The device-bound approval account was the one control in the release path that a stolen token could not satisfy directly. Had it still been in place, the attacker would have needed to use owner privileges to remove it, an action more likely to be noticed [2]. Relaxing it for convenience removed the only release factor that required physical possession. The attacker also deleted workflow run records with a third credential, which suggests that workflow records deletable by the identities they record are not a dependable detection source on their own [1].

Payload Behavior and Recovery Implications

The payload's design makes cleanup harder than a typical malicious package. Each wheel added `cook.pth`, `cook.py`, and `sitecustomize.py` to `site-packages`, and the Docker images carried byte-identical copies [4]. Python processes `.pth` files and `sitecustomize` during interpreter start-up. Microsoft found that the collector, which identifies itself as PRIVCOLLECT, runs in a daemon thread whenever Python starts in the affected environment [3]. StepSecurity and Datadog note that this includes running `pip` itself, so `pip uninstall` re-triggers the stealer [4][5]. The collector targets AI assistant and MCP configuration files, Git and package-manager credentials, cloud CLI credentials and metadata endpoints, Kubernetes service-account files, SSH and GPG keys,

browser credential stores, and other processes' environment variables. When `DATABASE_URL` is set, it also reads stored provider keys from the LiteLLM proxy database [3][4]. The data is encrypted with AES-256-CBC, with the AES key wrapped by an RSA-4096 public key, and posted to `litellm[.]dev/healthcheck`, a lookalike domain registered on October 13 [3][6].

The loader component deserves attention even though Microsoft did not confirm any second-stage execution [3]. StepSecurity reports that `sitecustomize.py` forks a detached daemon. Every hour, the daemon reads a hostname from a GitHub-hosted file, fetches code from `/codepy` on that host, and runs it with `exec()`. The GitHub account has since been deleted, but infected hosts keep polling. If someone re-registered the account name, they could deliver code to every host that is still infected [4]. The stealer writes its "already collected" marker only after a successful upload, so where exfiltration was blocked it will retry on every Python start [4]. For responders, this means that blocking the domain and uninstalling the package leaves a live, re-armable implant behind. This is consistent with LiteLLM's and Microsoft's recommendation to rebuild affected hosts rather than repair them in place [1][3].

Indicator Type	Value	Source
Affected versions	<code>litellm==1.107.0</code> (PyPI); Docker Hub and GHCR <code>v1.107.0</code> , <code>latest</code> , <code>main-latest</code> , <code>main-stable</code> ; Docker Hub tags pulled Oct 21, 11:46–14:20 UTC, GHCR tags pulled 11:46–13:51 UTC	[1]
Last known-good version	<code>litellm==1.106.2</code>	[1][5]
Files	<code>cook.pth</code> , <code>cook.py</code> , <code>sitecustomize.py</code> in <code>site-packages</code> ; <code>/tmp/remote_code_loader.lock</code> ; <code>/tmp/.pc_once_72129118c59f5636c0bb</code>	[1][4] [5]
Network	<code>litellm[.]dev</code> , <code>www.litellm[.]dev</code> , path <code>/healthcheck</code> ; any <code>/codepy</code> request	[3][4]
Source and access	Lookalike repo <code>Berri-AI/litellm</code> (commit <code>20535d8f</code>); IP <code>69.48.229[.]143</code> ; <code>ci-tmp-*</code> branches adding <code>.github/workflows/cache.yml</code>	[2][4] [6]
Image digest	<code>sha256:1bbcb1443bc962e45f5aad53fe3d2c30600e0379715fb6c664cdd3ac126c339e</code>	[1]

LiteLLM cautions that finding these files confirms exposure, but not finding them does not prove a host is clean [1]. LiteLLM states that enterprise customers who pulled from its private enterprise registry were not affected [1].

Recommendations

Immediate Actions

Organizations should first determine whether 1.107.0 reached any environment. They should search lockfiles, requirements files, `uv` and `pip` caches, internal mirrors, and container image digests. They should pay particular attention to unpinned `litellm>=` constraints and to deployments that track the floating Docker tags [3][4]. Any CI job that installed `litellm==1.107.0` or pulled the affected image tags should be included, whether directly during the October 21 exposure window or later from a cache or internal mirror. Every secret visible to that job, including other processes' environments, should be considered exposed [4]. Investigators should not run `python` or `pip` inside a suspect environment, since doing so runs the payload again [3][5].

For each affected host, responders should isolate the system, stop the loader daemon (the process holding `/tmp/remote_code_loader.lock`), and rebuild it from verified artifacts pinned to `litellm==1.106.2` or the `v1.106.2` image by digest [1][4]. From a known-clean machine, they should then rotate every credential the host could reach. That includes cloud instance and task roles, model-provider keys stored in the proxy database, LiteLLM master and salt keys, virtual keys, the `DATABASE_URL` password, SSH and GPG keys, CI secrets, AI-tool tokens, and browser sessions [1][3][4]. Teams should also block `litellm[.]dev` and review historical DNS and proxy logs for it and for `/codepy` requests, which would indicate attacker-supplied code ran [4]. LiteLLM also recommends checking cloud audit logs for identity-probing calls, such as AWS `GetCallerIdentity`, around the install time, and for use of rotated credentials from outside the organization's infrastructure [1].

Short-Term Mitigations

Organizations should review their own secret-exposure processes, using the May 27 commit as a test case. Every confirmed leak should start a revocation that has a named owner and is checked independently. The record should close only when the credential has been shown to fail authentication, not when the commit has been removed [9]. Teams should also inventory the credentials held by agent platforms, coding assistants, and MCP servers, and replace any human personal tokens with dedicated service identities. Where GitHub access is needed, a GitHub App or fine-grained token limited to the target repositories and permissions gives a far smaller blast radius than an organization owner's CLI token [2][18]. Agent platform APIs should never return stored credentials in response bodies. Static master keys for internal or alpha services should be rotated on a schedule and kept off the public internet.

For release pipelines, the lesson is to keep at least one approval factor that a stolen token cannot satisfy. Organizations should also treat changes to release workflow configuration as high-risk events that generate alerts. GitHub deployment environments support required reviewers, a "Prevent self-review" option, restrictions on which branches and tags may deploy, and a setting that disallows bypassing protection rules [10]. Release workflows should pin their source repository and reject forks or lookalike organizations. Teams should alert on branch-protection overrides, deleted workflow runs, and short-lived branches that add workflow files, because each of these appeared in this attack [1][2][6]. Workflow run and audit log data should go to storage the release maintainers cannot delete.

Strategic Considerations

The broader implication is that agentic tooling is changing where privileged human credentials live. Development teams now give agents the access they need to open pull requests, run builds, and call cloud APIs. Each of those platforms becomes a store of developer identity, which, as this incident illustrates, may have less mature authentication, logging, and revocation than the systems whose tokens it holds. Security programs should bring agent platforms into non-human identity governance on the same terms as CI/CD systems. That means a registered identity, least privilege, short-lived credentials, and revocation that can be verified [16].

Consumers of open-source AI infrastructure should also reconsider how they read supply chain signals. Signed artifacts and Trusted Publishing are valuable, but this incident shows that valid provenance can come from a pipeline the attacker controls. Pinning by hash or digest, adding new releases to internal mirrors only after a delay, and limiting outbound network access from build and inference hosts would each have reduced exposure in a release window that lasted less than three hours across PyPI and Docker [1][3][5]. Finally, organizations should expect high-value AI gateway projects to stay targets. LiteLLM has now had two pipeline compromises in seven months, so organizations should plan to apply the same scrutiny to every release.

CSA Resource Alignment

The most directly related CSA work is the companion research note *The Unrevoked Key: Agent Credential to LiteLLM Release Hijack* [13]. It reconstructs the same attack chain in detail and concludes that the lessons are "about credentials, not about the package." This note builds on that analysis. It concentrates on revocation as a verifiable control, on agent platforms as credential aggregators, and on consumer-side detection and recovery. Readers who need the full timeline and LiteLLM's account of how its explanation changed should consult both documents together.

CSA's analysis of the March 2026 campaign, *TeamPCP's LiteLLM Backdoor: New Data Shows 2,100+ Orgs Exposed* [14], is the closest precedent. That campaign began with an incomplete credential rotation at Aqua Security and spread through CI/CD into LiteLLM's PyPI releases. The October incident repeats the pattern at a different organization: a known exposure at LiteLLM that was never fully invalidated gave the attacker a way into the release path. Because the two failures occurred at different organizations, taken together they suggest that revocation follow-through is a recurring weak point across the AI supply chain, not a one-off error by a single project. CSA's *LiteLLM AI Gateway: KEV-Listed Attack Chain Enables Full Takeover* [15] and *LiteLLM's Default Admin Key: A Cloud Credential Vault* [17] cover the gateway side of the same risk. They explain why the stealer's targeting of the proxy database and master key gives a compromised LiteLLM host such a large blast radius. On the agent side, *AI Coding Agents: An Unaudited Supply Chain Node* [19] examines coding agents that hold developer credentials as a supply chain component in their own right, which is the risk the Agent Platform's owner token illustrates.

For control mapping, the CSA AI Controls Matrix (AICM) v1.1 [16] provides the governance structure for the recommendations above. Its Identity and Access Management domain covers least privilege and lifecycle management for the agent platform's token. Its Supply Chain Management, Transparency, and Accountability domain covers release-pipeline integrity and dependency pinning. Its Threat and Vulnerability Management domain covers detection and response for compromised packages. Organizations assessing AI gateway or agent platform vendors can use AICM controls to ask how stored credentials are scoped, how leaked secrets are revoked, and which release approvals require more than possession of a developer token.

References

- [1] LiteLLM (K. Dholakia, I. Jaffer, O. Jensen, N. Armstrong). "[Contained: Oct 21, 2026 LiteLLM release v1.107.0 Compromised](#)." LiteLLM Blog, October 21, 2026 (updated October 23, 2026).
- [2] LiteLLM (O. Jensen, M. Wang). "[The attack chain behind the v1.107.0 LiteLLM release](#)." LiteLLM Blog, October 23, 2026.
- [3] Microsoft Security Research. "[Inside the LiteLLM 1.107.0 supply chain compromise: Credential theft through Python startup hooks](#)." Microsoft Security Blog, October 23, 2026.
- [4] 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.
- [5] I. Kretz. "[LiteLLM supply chain compromise: What you need to know](#)." Datadog Security Labs, October 21, 2026.
- [6] R. Lakshmanan. "[Leaked LiteLLM Master Key Let Attackers Steal GitHub Tokens and Hijack Release Pipeline](#)." The Hacker News, October 24, 2026.
- [7] G. Korolevski, JFrog Security Research. "[LiteLLM Hijacked Again: Malicious 1.107.0 Wheels Steal Credentials and Load Remote Code](#)." JFrog, October 21, 2026.
- [8] BerriAI. "[\[SECURITY\] litellm 1.107.0 compromised \(PyPI removed; Docker latest/main-stable removed\) · Issue #47826](#)." GitHub, October 21, 2026.
- [9] GitHub. "[Removing sensitive data from a repository](#)." GitHub Docs, accessed October 26, 2026.
- [10] GitHub. "[Managing environments for deployment](#)." GitHub Docs, accessed October 26, 2026.
- [11] Python Packaging Authority. "[Trusted Publishers](#)." PyPI Docs, accessed October 26, 2026.
- [12] OpenSSF. "[SLSA Specification v1.0](#)." SLSA, accessed October 26, 2026.
- [13] Cloud Security Alliance. "[The Unrevoked Key: Agent Credential to LiteLLM Release Hijack](#)." CSA Labs, October 24, 2026.
- [14] Cloud Security Alliance. "[TeamPCP's LiteLLM Backdoor: New Data Shows 2,100+ Orgs Exposed](#)." CSA Labs, August 12, 2026.
- [15] Cloud Security Alliance. "[LiteLLM AI Gateway: KEV-Listed Attack Chain Enables Full Takeover](#)." CSA Labs, June 17, 2026.
- [16] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1](#)." Cloud Security Alliance, accessed October 26, 2026.
- [17] Cloud Security Alliance. "[LiteLLM's Default Admin Key: A Cloud Credential Vault](#)." CSA Labs, September 10, 2026.
- [18] GitHub. "[Deciding when to build a GitHub App](#)." GitHub Docs, accessed October 26, 2026.
- [19] Cloud Security Alliance. "[AI Coding Agents: An Unaudited Supply Chain Node](#)." CSA Labs, July 8, 2026.