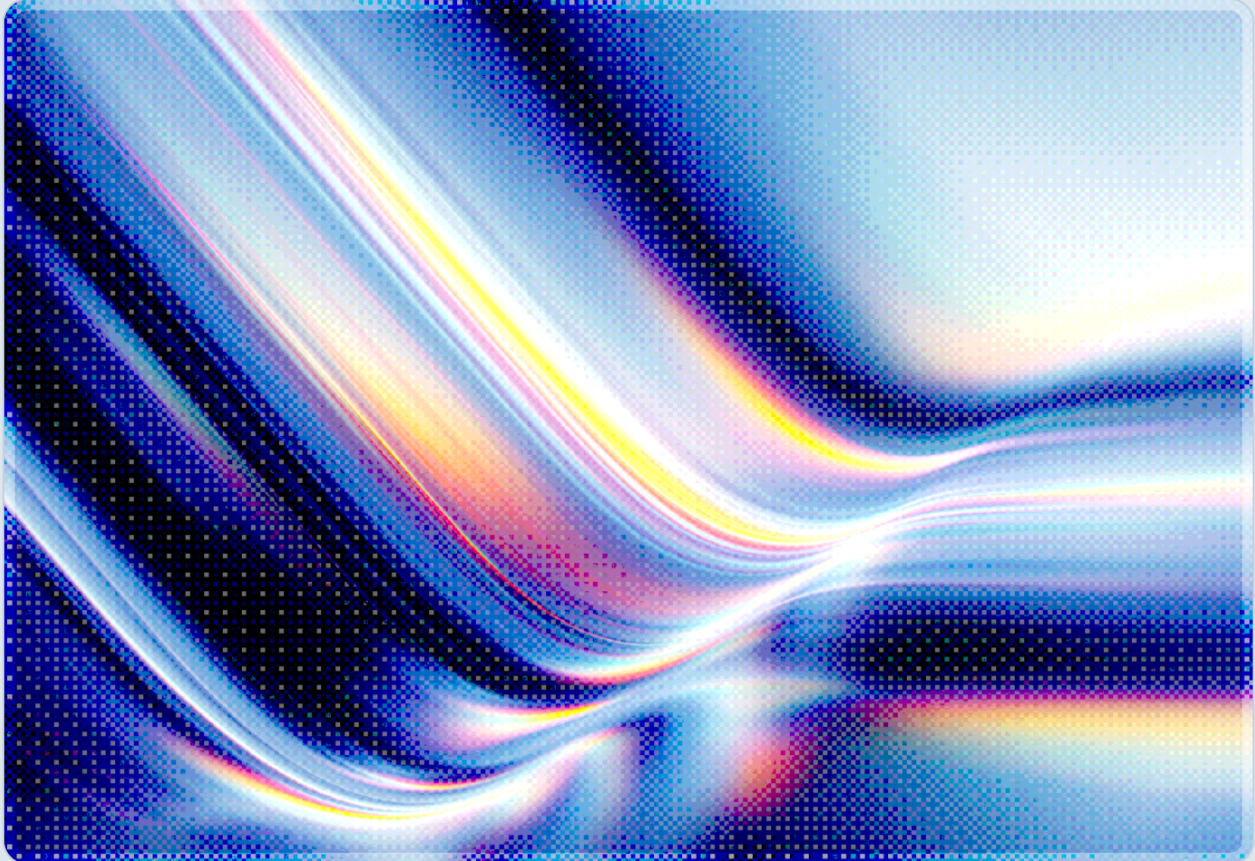


LiteLLM 1.107.0: When Model-Gateway Logs Become the Credential Store

Security Implications and Guidance for AI Gateway Operators

2026-10-23

 AI-assisted Rapid Research



CSAI

cloud
CSA 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, malicious builds of the LiteLLM AI gateway, version 1.107.0, were published to PyPI, Docker Hub, and GitHub Container Registry. The builds carried a credential stealer and a remote-code loader that run every time Python starts [1][2]. LiteLLM reports that the attacker drove its own release pipeline using stolen GitHub CLI OAuth tokens belonging to three of its engineers. What sets this incident apart is LiteLLM's explanation of how those tokens were obtained. The company believes that an internet-exposed internal LiteLLM deployment had recorded request and response bodies from Claude Code sessions, and that the gateway's logs therefore held developers' GitHub credentials, which have nothing to do with model access [1][5]. In LiteLLM's account, those credentials became the foothold for a software supply chain attack. How the attacker first reached the internal deployment remains unconfirmed [1].

The immediate priority is the hosts that ran the bad release. If `litellm` 1.107.0 ran on a host with outbound network access, responders should assume exfiltration succeeded. They should delete the dropped files without starting Python, rebuild from clean images pinned to 1.106.2, and rotate every credential the host could reach, working from a clean machine [1][2]. Exposure through logs, however, does not depend on having installed the bad release at all. Any LiteLLM deployment that stores request or response content for coding-agent traffic should be audited for embedded tokens [8]. More broadly, logged gateway traffic deserves the same controls as a secret store. An AI gateway that records prompts and completions for agentic tools holds whatever secrets those agents read, print, or echo, and those can include source-control, cloud, and package-registry credentials.

The attacker's later activity and the project's history add further context. Wiz observed stolen AWS role credentials being validated and then used to enumerate Amazon Bedrock models about four and a half hours after theft, a pattern consistent with LLMjacking [4]. This is also LiteLLM's second supply chain compromise of 2026, following a March incident [6]. Socket has said that the public evidence reviewed so far does not establish a link between the two [5].

Background

LiteLLM is an open-source library and proxy server from BerriAI. It gives applications a single OpenAI-compatible interface to many model providers. In proxy mode it acts as a central AI gateway: it stores upstream provider keys, issues virtual keys to teams and applications, enforces budgets, and records spend and usage. Socket estimates the package's weekly downloads at about 23 million, as reported by Help Net Security [6]. That scale, and the gateway's role as a broker of high-value provider credentials, make LiteLLM an attractive target, and October's incident is the project's second supply chain compromise of 2026. Help Net Security reports that the March 2026 attack produced a 153GB stolen archive, which Hudson Rock linked to 2,488 corporate domains [6].

LiteLLM's incident report gives a precise timeline for the October compromise [1]. At 10:08 UTC on October 21, GitHub branch protection rejected an unauthorized change to a private release workflow. Five seconds later, a privileged developer credential overrode those protections and applied the change. At 10:46 UTC the modified workflow began a staging run against `Berri-AI/litellm`, a hyphenated lookalike of the legitimate `BerriAI/litellm` repository. A second developer credential satisfied the production approval gate at 11:11 UTC. Seven malicious wheels reached PyPI between 11:39 and 11:40 UTC, and container images followed at 11:46 UTC. At 11:53 UTC, a third developer account deleted production and staging workflow records, which limited the forensic evidence available [1]. Wiz adds that the attacker pushed ten commits to the lookalike repository between 08:56 and 10:46 UTC, nine of them signed by GitHub [4].

LiteLLM describes the root cause as a chain of control failures rather than a single defect [1]. Two days before the compromise, on Monday, October 19, LiteLLM had temporarily expanded release permissions so that two engineers could publish daily bugfix backports [1]. The attacker held three sets of GitHub CLI credentials, and one of them belonged to a GitHub organization owner. That owner token allowed branch protections to be disabled and the release workflow to be redirected to a different source repository. LiteLLM then described how it believes the tokens were obtained. In its words, "we have observed specific situations in which Claude Code will embed a user's GitHub CLI OAuth token into a response body," and "in situations where response bodies are logged, these tokens may appear in LiteLLM logs" [1][8]. The company found an "innocuously named" user account on an internal LiteLLM deployment exposed to the public internet. It suspects, but has not confirmed, that the attacker entered that deployment through a recently patched vulnerability [1][5]. In the coverage reviewed for this note, Anthropic had not commented publicly on the token-embedding behavior [5][7].

The table below summarizes the affected artifacts and exposure windows as reported by LiteLLM. Wiz reports slightly different removal times, about 13:20 UTC for PyPI and 13:53 UTC for container images [4]. Those are earlier than LiteLLM's figures for PyPI and Docker Hub, but slightly later than its GHCR

figure. Responders should therefore use the widest window for each channel when they scope exposure.

Distribution channel	Affected artifacts	Exposure window (UTC, Oct 21)
PyPI	<code>litellm==1.107.0</code> , all seven wheels; the sdist contains a tampered <code>pyproject.toml</code> but no payload files [2]	11:39:51 – 13:23 [1]
Docker Hub	Tags <code>v1.107.0</code> , <code>latest</code> , <code>main-latest</code> , <code>main-stable</code> (index <code>sha256:1bbcb144...</code>)	11:46:56 – 14:20 [1]
GHCR	Same four tags on <code>litellm</code> , <code>litellm-database</code> , <code>litellm-non_root</code>	11:46:56 – by 13:51 [1]
Source	The official <code>v1.107.0</code> git tag does not contain the payload [1][2]	Not affected

Wiz reports about 49,000 downloads of the malicious version, based on ClickPy data [4]. Download counts include CI caches and mirrors, so they overstate unique victims. They also omit container pulls, so they understate exposure through the `latest` tag. Enterprise customers who pulled from LiteLLM's private registry were not affected [1].

Security Analysis

The incident combines two distinct problems that deserve separate treatment. The first is the malware itself, which was engineered to run on any Python startup and to target the credential stores that LiteLLM deployments manage. The second, and in CSA's view the more instructive, is the path LiteLLM believes the attacker used to obtain release authority in the first place. The subsections below examine the payload, the role of gateway logs, the release-pipeline weaknesses the stolen tokens exploited, and the attacker's activity after exfiltration.

The payload: Python startup hooks and a gateway-aware stealer

The Upwind security research team recovered the payload by static decoding and documented it in the public GitHub issue that first raised the alarm [2]. Each malicious wheel added three files, `cook.pth`, `cook.py`, and `sitecustomize.py`, through a maturin `data = "cook"` directive that is absent from the tagged source tree. Pip installs these files at the root of site-packages. Python processes `.pth` files and `sitecustomize.py` automatically at interpreter startup, so the malware runs whenever any Python process starts in the affected environment, whether or not LiteLLM is imported. That includes `pip` itself. Because `pip` is a Python process, uninstalling or reinstalling with `pip` executes the payload one more time, which is why both LiteLLM and Upwind warn against starting remediation with `pip` [1][2]. Wiz describes the combination of `.pth` and `sitecustomize.py` persistence as appearing to be the first documented case of its kind in the wild [4].

The stealer, which calls itself `privcollect`, covers far more than AI keys. Upwind's analysis lists host and network reconnaissance, decrypted browser passwords and cookies, SSH keys, shell and database histories, and AWS, Azure, GCP, Kubernetes, Docker, and GitHub CLI credentials. It also collects package-registry tokens, `.env` files, private keys, Vault and Terraform secrets, configurations for AI coding tools, and the environment variables of every readable process [2]. It queries cloud instance metadata services and validates AWS role credentials with STS `GetCallerIdentity`. Of particular relevance to gateway operators, when `DATABASE_URL` is set the stealer runs Prisma to dump `LiteLLM_CredentialsTable` and `LiteLLM_ProxyModelTable`, which together hold every upstream provider key the proxy manages [2]. The collected data is encrypted with AES-256-CBC under an RSA-4096-wrapped key and POSTed to `http://www.litellm.dev/healthcheck`. That domain was registered on October 13 and is not the project's legitimate `litellm.ai` domain [2]. A second stage polls a GitHub dead-drop file every hour for a domain from which to download and `exec()` further code. That stage was never observed armed, and GitHub has since removed the repository [2][4].

Gateway logs as an unplanned credential store

In CSA's assessment, the incident's most consequential lesson for AI infrastructure is the initial-access path LiteLLM describes, more than the malware. AI gateways have usually been threat-modeled around the provider keys they store deliberately. This incident points to a second, unplanned credential store: the traffic log. When a coding agent such as Claude Code is routed through a gateway, every request carries the agent's working context, and every response carries the model's output. That context can include tool results, terminal output, file contents, and environment details. If an agent reads or prints a credential while working, that credential can end up in a request or response body. One plausible route is a command

that displays an authentication token, but LiteLLM has not published the specific triggering conditions, so that mechanism remains an inference. However it happens, a gateway that records message content then holds the credential, along with whatever access control, retention, and replication policy applies to its logs.

In LiteLLM's case, request and response content is not stored in spend logs by default. Operators opt in with `store_prompts_in_spend_logs`, either in configuration or through an Admin UI toggle that takes effect immediately and overrides the config file [10]. A separate setting, `turn_off_message_logging`, keeps message and response content away from logging callbacks while still tracking spend [9]. These controls exist, but they are opt-in, they can be changed at runtime from the UI, and their interaction has not always been reliable. A public issue against version 1.79.0 reported that with both settings enabled, responses were redacted but request messages persisted in the `proxy_server_request` metadata field [11]. The issue has since been closed, but CSA has not verified which releases contain a fix, so operators should confirm the behavior on their own version. Because operators often enable content logging deliberately for debugging, evaluation, or cost attribution, these settings cannot be relied on as a protective default.

Logs also make poor custodians of secrets. In many organizations they are readable by a wider group than the secrets vault, they are retained for months to support audits and cost analysis, and they are shipped to downstream observability, tracing, and evaluation platforms through callbacks. A token that lands in a gateway's spend-log table can be copied into a database backup, an analytics export, and a third-party tracing service shortly after it is written. As a result, the blast radius of one logged credential depends on the security of every system that ingests the log, not only the gateway. The credentials at risk also extend well beyond model access. In this case they were source-control tokens with release authority over a widely used package.

This exposure path reaches beyond the population that installed the malicious release. A log-borne token can be exposed even when a deployment never ran version 1.107.0, and the remediation LiteLLM initially published did not tell operators who route Claude Code through LiteLLM and record response bodies to audit their logs or rotate tokens. An automated reviewer on LiteLLM's documentation pull request was the first to note this gap [8]. CSA concludes that the population potentially affected by log-borne token exposure includes any gateway deployment that recorded coding-agent traffic, a group that differs from, and may be larger than, the set of hosts that installed the bad build.

Release-pipeline weaknesses the tokens exploited

The stolen tokens were dangerous because of the authority attached to them. Branch protection did reject the first unauthorized change, but an organization-owner credential could override it within seconds [1]. One stolen identity disabled the control, a second approved production, and a third erased workflow history. The approval gate therefore separated duties between people but not between compromised identities. Provenance offered only partial help. Datadog notes that build provenance named the lookalike fork as the source [3], and Upwind indicates that this SLSA provenance was attached to the container images, while LiteLLM had never published PyPI attestations for its wheels [2]. Users installing from PyPI therefore had no straightforward way to tell a tampered build from a legitimate one. Had provenance been verified at install time, the hyphenated source repository could have stood out as an anomaly before deployment.

The temporary expansion of release permissions two days before the compromise is part of the same picture. LiteLLM lists this expansion among the contributing control failures [1]. The tokens that were abused included an organization-owner token, which would have carried release authority regardless, so the note does not establish that the attacker relied on the widened permissions. The episode is nonetheless a reminder that time-boxed privilege grants need explicit expiry and review, even when a given grant may not have been the path an attacker used.

Post-compromise activity

Wiz reported that, starting about four and a half hours after exfiltration, the attacker tested AWS instance-role credentials stolen from compromised CI runners and Kubernetes nodes. The testing came from two IP addresses, `168.100.8[.]57` and `165.227.201[.]8` [4]. Each credential was checked with `sts:GetCallerIdentity`, followed by `bedrock:ListFoundationModels` and a model-listing call. Wiz characterizes this sequence as consistent with LLMjacking, the resale or abuse of stolen cloud access to run generative AI models. In a small number of environments the attacker also enumerated IAM resources [4]. These cloud credentials came from a different path than the GitHub tokens: they were stolen by the `privcollect` payload on hosts that installed the malicious package, not harvested from gateway logs. Taken together, the two paths show AI infrastructure serving as both the source and the target of the theft, with cloud credentials stolen from hosts running the malicious LiteLLM build used within hours in an attempt to obtain AI compute [4].

Recommendations

The recommendations below are organized by time horizon. Immediate actions address hosts that ran the malicious release and gateways that may hold log-borne tokens. Short-term mitigations focus on logging configuration, developer credentials, and release integrity, and the strategic considerations address how organizations classify and govern AI gateways as a category of infrastructure.

Immediate Actions

Organizations should first determine whether `litellm==1.107.0` or any of the four affected container tags were pulled during the exposure windows. The checks should cover developer laptops, CI runners, and transitive dependencies of other Python tools. Upwind and Datadog publish indicators that can be searched across fleets, summarized in the table below [2][3].

Indicator type	Indicator
Dropped files	<code>cook.pth</code> , <code>cook.py</code> , and <code>sitecustomize.py</code> at the root of any site-packages directory, plus a <code>litellm-1.107.0.dist-info</code> directory
Marker and lock files	Run-once marker <code>/tmp/.pc_once_72129118c59f5636c0bb</code> , written only after a successful exfiltration POST; lock file <code>/tmp/remote_code_loader.lock</code>
Process artifacts	Environment variable <code>PRIVCOLLECT_ACTIVE=1</code> , or an orphaned <code>python</code> process with working directory <code>/</code>
Network	Outbound connections to <code>litellm[.]dev</code> , or HTTP requests carrying the header <code>X-Token: XWiA8gZNa7NcX2XRoAEb</code>

On affected hosts, responders should remove the dropped files and their `__pycache__` entries from the shell before starting any Python process. They should then kill the daemon holding the lock file and rebuild environments from scratch pinned to `1.106.2` [1][2]. A missing marker file does not show that exfiltration failed, because container restarts and tmp cleanup remove it [1]. Credential rotation should be done from a clean machine. It should cover instance and task IAM roles, every provider key in the LiteLLM proxy database, master and virtual keys, the `DATABASE_URL` password, and any secret present in the

environment of any process on the host [1][2]. Responders should also search CloudTrail for `GetCallerIdentity` calls from affected roles around install time, and for Bedrock enumeration from the IP addresses Wiz reported [1][4].

Every LiteLLM deployment, whether or not it installed 1.107.0, should immediately check whether request or response content has been stored for coding-agent traffic. The same reasoning applies to any AI gateway, proxy, or tracing layer that records such traffic. Where content has been stored, operators should scan it for credential patterns and revoke anything found. GitHub tokens are the immediate concern, including the `gho_` prefix used by OAuth tokens such as GitHub CLI tokens as well as the `ghp_`, `github_pat_`, `ghu_`, and `ghs_` prefixes, but cloud and registry keys belong in the same scan. The table below lists the locations worth checking.

Log location	Why it matters	Action
Spend-log table (<code>store_prompts_in_spend_logs</code> enabled)	Holds request/response content in the proxy database [10]	Scan for token patterns, then purge or redact
<code>proxy_server_request</code> metadata	Reported (v1.79.0, Issue #16336, now closed) to retain request messages even with redaction enabled; fix version unverified [11]	Include in scans regardless of settings
Logging callbacks (tracing, observability, evaluation platforms)	Receive message content unless <code>turn_off_message_logging</code> is set [9]	Scan downstream stores and request vendor-side deletion
Database backups and analytics exports	Replicate log content outside the gateway's access controls	Scope retention, then scan or expire

Short-Term Mitigations

Over the coming weeks, gateway operators should make content logging an explicit, reviewed exception rather than a convenience setting. Where prompts and completions must be retained, a secret-scanning and redaction stage should run before persistence and before callbacks forward the data. Access to stored content should be restricted to the same small group that can read the gateway's credential store. Because the LiteLLM Admin UI can change logging behavior at runtime [10], configuration drift monitoring should alert whenever content storage is enabled. Gateway administration interfaces should never be reachable from the public internet. The internal deployment in this incident was internet-exposed [1], and the uncertainty about how it was entered should itself prompt a review of internet-facing AI infrastructure.

Developer credentials used alongside coding agents also need attention. GitHub CLI OAuth tokens are typically long-lived and broadly scoped; the CLI documents `repo`, `read:org`, and `gist` as the minimum scopes it requires [17]. Organizations should prefer fine-grained, short-lived tokens for agent sessions and keep organization-owner credentials off developer workstations that run agents. Release workflows should be restricted to approved source repositories and branches [3]. Projects that publish to PyPI should adopt Trusted Publishing with attestations [2][12], and consumers should verify provenance before promoting new releases into production. Pinning dependencies by version and container images by digest, rather than tracking `latest` or `main-stable`, could have kept automated deployments that track those tags out of the exposure window.

Strategic Considerations

The broader lesson is architectural. As coding agents and other agentic tools are routed through centralized gateways for cost control and governance, those gateways see an increasingly complete record of developer activity, and in practice a record of the developers' secrets as well. Security architects should classify gateway traffic logs alongside credential vaults and source code. They should apply data-minimization by default and assume that any secret an agent can see may appear in a prompt or completion. Agent vendors also have a part to play, since tooling that keeps credentials out of model context, or masks them before transmission, reduces what gateways and providers can inadvertently retain.

Organizations should also reassess how much trust they place in fast-moving AI infrastructure dependencies. LiteLLM has now been compromised through its supply chain twice in a single year [6]. Given what AI gateways broker, treating them as tier-zero infrastructure is warranted. In practice that means staged rollouts, provenance verification, isolated runtime identities, and allowlist-based egress restrictions, which would likely have blocked the newly registered exfiltration domain.

CSA Resource Alignment

CSA's **AI Controls Matrix (AICM) v1.1** provides the most direct control mapping for this incident [13]. The Logging and Monitoring domain covers protecting log integrity and confidentiality and restricting access to audit data. This incident shows those controls must extend to AI gateway traffic logs that capture message content. The Identity and Access Management and Cryptography, Encryption and Key Management domains address the lifecycle, scoping, and rotation of the provider keys and developer tokens that were exposed. The Supply Chain Management, Transparency, and Accountability domain applies to provenance verification and dependency pinning for AI middleware such as LiteLLM. Organizations assessing their gateways against AICM should explicitly bring prompt and completion stores into scope for their logging and data-protection controls.

CSA's **Identity and Access Gaps in the Age of Autonomous AI** examines how AI agents operate inside enterprise systems, often with access inherited from the humans who run them rather than distinct, scoped identities [14]. A coding agent working under a developer's GitHub CLI session, whose activity carries that developer's long-lived token into a logged gateway, is a concrete example of the inherited-access problem the report describes. Its emphasis on identity-bound, short-lived, per-task access for agents corresponds directly to the token hygiene recommended above.

CSA's **Defining Non-Human Identity** and **The State of Non-Human Identity and AI Security** frame the provider keys, virtual keys, OAuth tokens, and cloud role credentials in this incident as non-human identities that need inventory, ownership, rotation, and vaulting [15][16]. Their governance guidance addresses the question this incident raises most sharply: where an organization's credentials actually reside. An NHI inventory that covers vaults and configuration files but not AI gateway logs would have missed the store that LiteLLM believes enabled this attack.

References

- [1] LiteLLM. "[Contained: Oct 21, 2026 LiteLLM release v1.107.0 Compromised](#)." LiteLLM Blog, October 21, 2026.
- [2] Upwind Security Research (reporter). "[\[SECURITY\] litellm 1.107.0 compromised: cook.pth credential stealer + sitecustomize RCE loader, built from lookalike repo Berri-AI/litellm \(Issue #47826\)](#)." GitHub, October 21, 2026.
- [3] Kretz, Ian. "[LiteLLM supply chain compromise: What you need to know](#)." Datadog Security Labs, October 21, 2026.
- [4] Read, Benjamin and McCarthy, Rami. "[LiteLLM Compromised, Steals Credentials](#)." Wiz, October 21, 2026.
- [5] Lakshmanan, Ravie. "[LiteLLM Suspects GitHub Token Leak in AI Gateway Logs Behind Malicious Package Release](#)." The Hacker News, October 22, 2026.
- [6] Markovic, Sinisa. "[LiteLLM hit by second supply chain attack this year, users urged to rotate credentials](#)." Help Net Security, October 22, 2026.
- [7] Constantin, Ana Maria. "[LiteLLM says hackers poisoned it with GitHub tokens leaked via Claude Code](#)." The Next Web, October 23, 2026.
- [8] BerriAI. "[docs\(blog\): add 20:30 UTC status update to Oct 21 incident post \(Pull Request #2539\)](#)." GitHub, October 21, 2026.
- [9] LiteLLM. "[Logging](#)." LiteLLM Documentation, accessed October 23, 2026.
- [10] LiteLLM. "[UI Logs](#)." LiteLLM Documentation, accessed October 23, 2026.
- [11] BerriAI. "[\[Bug\]: turn off message logging Does Not Redact Request Messages in proxy server request Field When Stored to Database \(Issue #16336\)](#)." GitHub, accessed October 23, 2026.
- [12] Python Packaging Authority. "[Trusted Publishers](#)." PyPI Documentation, accessed October 23, 2026.
- [13] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1](#)." Cloud Security Alliance, 2026.
- [14] Cloud Security Alliance. "[Identity and Access Gaps in the Age of Autonomous AI](#)." Cloud Security Alliance, March 2026.

- [15] Cloud Security Alliance. "[Defining Non-Human Identity](#)." Cloud Security Alliance, July 2026.
- [16] Cloud Security Alliance. "[The State of Non-Human Identity and AI Security](#)." Cloud Security Alliance, January 2026.
- [17] GitHub. "[gh auth login](#)." GitHub CLI Manual, accessed October 23, 2026.