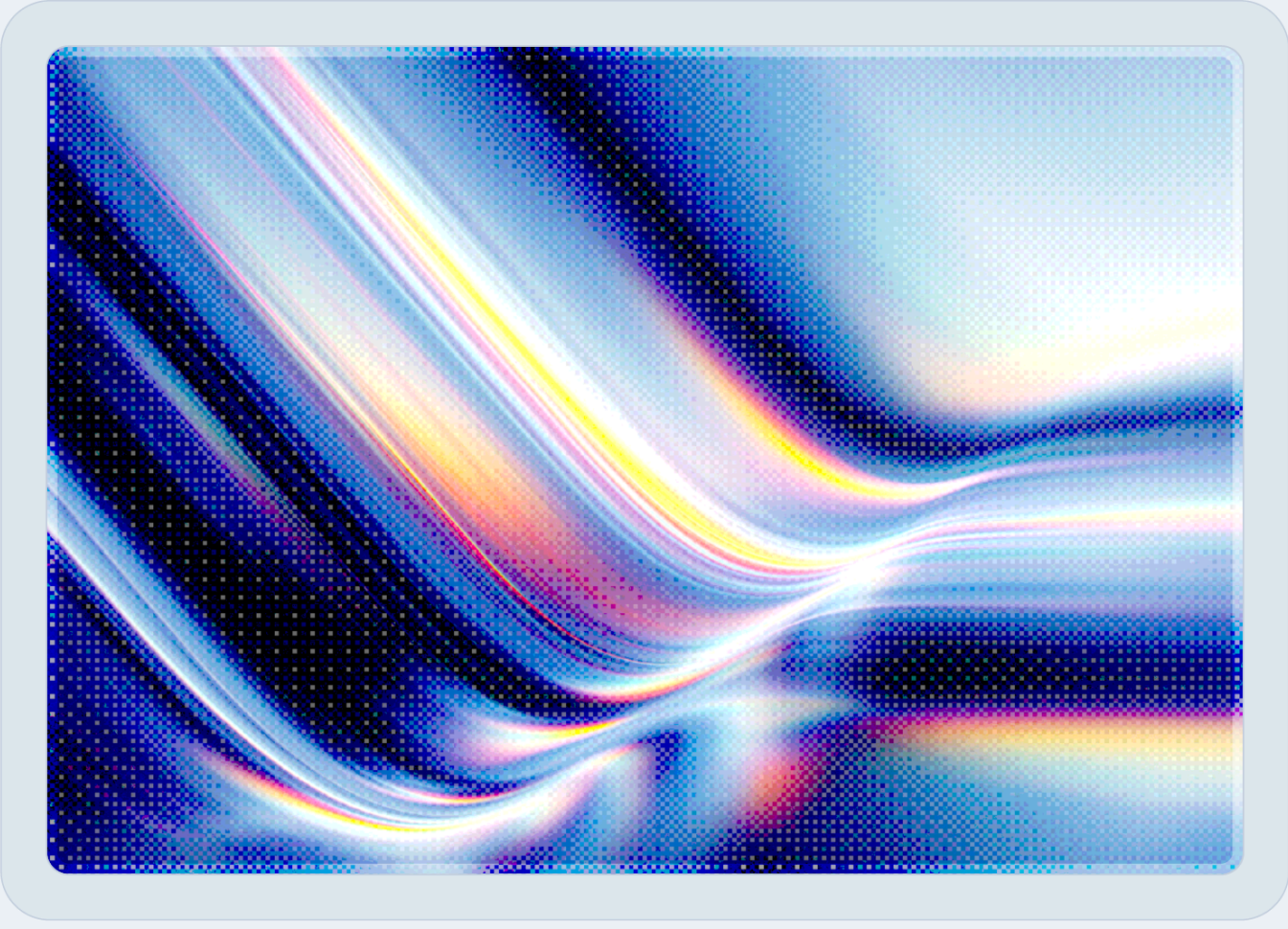
CSAI Foundation | Cloud Security Alliance

Guardrails as an Execution Channel: LiteLLM CVE-2026-40217 Enters KEV

A custom-code guardrail sandbox escape in the LiteLLM AI gateway is now confirmed exploited

2026-11-20

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

On November 18, 2026, the U.S. Cybersecurity and Infrastructure Security Agency (CISA) added CVE-2026-40217 to its Known Exploited Vulnerabilities (KEV) catalog. CISA describes it as an "unprotected alternate channel vulnerability" in BerriAI's LiteLLM that "could allow remote attackers to execute arbitrary code via bytecode rewriting at the /guardrails/test_custom_code URI." Federal civilian agencies must remediate it by December 2, 2026 [1]. The vulnerability sits in a feature that was built for safety. LiteLLM lets administrators write custom guardrails in Python and test them through an API endpoint. The sandbox that confined that code could be escaped, so the code ran inside the proxy process. In the default Docker image, that process runs as root [2][3].

This is the fourth LiteLLM vulnerability to enter the KEV catalog in 2026, after CVE-2026-42208 (SQL injection, added May 8), CVE-2026-42271 (command injection, added June 8), and CVE-2026-59822 (MCP authentication bypass, added September 2) [1]. The pattern supports a conclusion CSA has drawn in earlier research: organizations should manage AI gateways as Tier-0 infrastructure, applying the same patching, access-control, and monitoring standards they apply to identity providers and secrets stores [12][13].

Read on its own, the vulnerability requires proxy-administrator credentials [2]. That reading understates the practical risk. Obsidian Security published a chain in which CVE-2026-47101 and CVE-2026-47102 let a default low-privilege user become `proxy_admin`, and CVE-2026-40217 then turns that role into code execution [4]. Google Threat Intelligence Group (GTIG) and Mandiant report that a threat actor used CVE-2026-40217 in LLMjacking intrusions to turn limited-access "Virtual Keys" into full administrative credentials. Mandiant's broader LLMjacking investigations, of which the LiteLLM case was one, spanned victims in financial services, retail, transportation, and technology [5]. The two GTIG publications describe the CVE's mechanism differently from the vendor advisory, and this note covers that discrepancy below.

The patched release is LiteLLM 1.83.10 according to the advisory metadata and 1.83.11 according to the advisory text. Version 1.83.14-stable is the first release that fixes the full Obsidian chain [2][4]. CISA's catalog entry calls the flaw "remote," but that word describes the network path rather than an absence of authentication, so readers of the catalog alone should not misjudge the privilege requirement [8]. Teams upgrading now must also avoid the malicious `litellm==1.107.0` release and the Docker tags that were poisoned on October 21, 2026 [10][13]. Teams that cannot upgrade immediately should block

`POST /guardrails/test_custom_code` at the reverse proxy, audit for unauthorized `proxy_admin` accounts and guardrail definitions, and rotate the credentials the gateway holds if they find any sign of misuse [2][3][4].

Background

LiteLLM is an open-source Python SDK and proxy server maintained by BerriAI. It puts one OpenAI-compatible API in front of more than 100 model providers [3]. Enterprises deploy the proxy as an "AI gateway" that centralizes routing, spend tracking, rate limiting, logging, and policy enforcement for applications and agents that call large language models. The gateway holds upstream provider API keys and issues its own scoped "Virtual Keys" to internal consumers. Compromising it therefore exposes both a credential store and a control point over model traffic. CSA's prior LiteLLM research notes have traced this concentration of value through a SQL injection, a command injection chain, two supply chain compromises (the March 2026 TeamPCP poisoning of versions 1.82.7 and 1.82.8, and the October 2026 compromise of version 1.107.0), and post-compromise abuse of the gateway's own callback system [12][13][14][16].

Guardrails are one of the gateway's policy features. Operators can attach checks that inspect prompts and responses, and LiteLLM supports guardrails written as custom Python code. To help administrators develop them, the proxy exposes `POST /guardrails/test_custom_code`, which runs submitted code against sample input and returns the result [2][3]. Executing operator-supplied code is a legitimate design choice. The security of the feature then depends entirely on the isolation around that execution, and in this case the isolation was a hand-built filter rather than an operating-system or interpreter boundary.

X41 D-Sec found the flaw in February 2026 and published advisory X41-2026-001 on April 8, 2026, after a coordinated-disclosure period that went through the distros mailing list [3]. CVE-2026-40217 was published on April 10, 2026 [17]. LiteLLM 1.83.10-stable shipped in late April and replaced the sandbox with RestrictedPython [2][4][8]. Obsidian Security, which credits the test-endpoint path to X41, independently found a variant of the same flaw and used it as the final link of its privilege-escalation chain. Obsidian tracks that variant under the same CVE: the production guardrail-creation paths (`POST /guardrails` and `PUT /guardrails/{id}`), which ran custom code through `exec()` with Python builtins still reachable [4]. The vulnerability was listed in KEV roughly seven months after a patch became available. KEV listings track evidence of exploitation rather than patch adoption, and the timing of GTIG's reporting suggests the exploitation was detected recently. Even so, confirmed exploitation this long after a fix suggests that a meaningful number of self-hosted gateways remained unpatched.

Security Analysis

How the Sandbox Failed

X41's advisory explains the root cause. The test endpoint protected itself with regex-based filtering of the submitted Python source text, and it denied dangerous names such as `__globals__`, `__builtins__`, and `__import__` [3]. Python lets code build identifiers at runtime and change compiled code objects, so a filter that only examines source text cannot tell what the code will do when it runs. X41's proof of concept builds the forbidden names through string concatenation. It then rewrites a generator's code object (`code.replace(co_names=...)`) to swap in those names, and uses the result to reach the real `__import__` function through a function's globals. From there it runs operating system commands [3]. Ostorlab's analysis reaches the same conclusion: the old sandbox "filtered the text of a Python program but could not reliably restrict what it did at runtime" [8].

The general lesson reaches well beyond LiteLLM. Sandboxes for dynamic languages have a long record of bypasses, because an interpreter offers far more ways to reach a dangerous capability than a list of forbidden strings can cover. RestrictedPython, the replacement LiteLLM adopted, works at the compilation stage instead of on raw text, which closes the class of bypass X41 demonstrated [2][8]. Its maintainers state, however, that RestrictedPython "is not a sandbox system or a secured environment" [18], and the library itself has had sandbox-escape vulnerabilities, such as CVE-2023-37271, which reached the frame stack through generators [19]. It should therefore be treated as defense in depth, not as an isolation boundary. Beyond that, any feature that runs user-supplied code inside the gateway process puts that code in the same address space as provider keys, database credentials, and routing configuration. The safest design runs such code in a separate, unprivileged process or container with no access to gateway secrets. CSA observed the same architectural weakness, in which user-influenced code runs inside an AI gateway behind an insufficient sandbox, through a different mechanism (prompt-template injection) in the GitLab AI Gateway sandbox escape it analyzed in October 2026 [15].

Who Can Reach the Endpoint

The sources disagree on the privilege requirement, and defenders should understand why. The GitHub advisory states that "reaching the endpoint requires a proxy-admin credential in default configurations," and its CVSS v4.0 vector uses PR:H, for a score of 7.5 [2]. X41's own vector uses PR:L (CVSS 8.7), and secondary reporting cites a CVSS v3.1 score of 8.8 [3][7]. Obsidian's timeline explains part of the gap. A

February 2026 change added `PROXY_ADMIN` checks to the guardrail endpoints, so versions after v1.82.0-stable require administrator access for direct exploitation [4]. CISA's description uses the word "remote." Ostorlab notes that this describes the network path, not the absence of authentication [8].

The privilege requirement offers less protection than it seems, for three reasons. First, Obsidian's CVE-2026-47101 let non-admin users mint keys with unrestricted `allowed_routes`, and CVE-2026-47102 let users set their own `user_role` to `proxy_admin`. Together they produce a path from a default `internal_user` to administrator, and CVE-2026-40217 completes it to code execution [4]. Second, the LiteLLM master key is a long-lived static secret that can easily spread across configuration files, CI pipelines, and environment variables. CSA's analysis of the 1.107.0 release compromise documented one such case, in which a single unrevoked master key for BerriAI's own LiteLLM Agent Platform deployment enabled a much larger breach [13]. Third, GTIG reports that the observed attackers started from Virtual Keys they had acquired from another threat actor, which is the low-privilege starting point the Obsidian chain needs [5].

Exploitation in the Wild

CISA's listing confirms exploitation evidence but names no victims or actors. The entry marks known ransomware use as "Unknown" [1]. The most detailed public account comes from Google. GTIG and Mandiant's November 2 report on LLMjacking describes a follow-on threat cluster that obtained "pre-validated Virtual Keys" from the actor that gained initial access. The cluster then exploited the victim's LiteLLM gateway "to elevate standard Virtual Keys into full PROXY_ADMIN master administrative credentials" [5]. The same report describes LLMjacking intrusions where fraudulent model consumption exceeded $100,000 in a single day, with cumulative charges at some organizations reaching $1 million before access was revoked [5]. Those figures cover the broader LLMjacking activity Mandiant observed. The report does not attribute them to the LiteLLM exploitation alone.

Defenders should note an inconsistency in the public record. GTIG's November 2 report describes CVE-2026-40217 as "a user ID rebind vulnerability in the /key/regenerate endpoint" [5]. GTIG's November 16 AI Threat Tracker calls it "a sandbox escape in LiteLLM" that elevates Virtual Keys to administrative credentials [6]. The vendor advisory, X41, Obsidian, and CISA all describe CVE-2026-40217 as the guardrail code-execution flaw [1][2][3][4]. Obsidian assigns key-route abuse on endpoints including `/key/regenerate` to CVE-2026-47101, and self-promotion to `proxy_admin` to CVE-2026-47102 [4]. One plausible reading is that the observed intrusions used the full chain and that GTIG's shorthand merged its steps under one identifier. This is an inference, and CSA has not confirmed it with Google; CSA will update this note if Google clarifies the attribution. The practical consequence does not depend on which reading is correct. Organizations should fix and hunt for all three CVEs together, not treat CVE-2026-40217 alone.

Impact of Successful Exploitation

Code execution in the LiteLLM proxy process gives an attacker everything the gateway holds and controls. That includes the upstream provider API keys, the master key, the database connection string and through it every Virtual Key and spend record, and the ability to change how traffic is routed. Because the default container runs as root, the attacker also controls the container and may be able to reach mounted cloud credentials or the orchestration environment [2][3]. CSA's August 2026 analysis of "LLM Heist" showed that an attacker with administrative control can redirect model traffic, harvest resolved provider keys, and inject forged tool calls into responses that downstream agents will execute. All of that can be done with the gateway's own legitimate management API and callback hooks [14]. Because guardrail and callback definitions are stored in the proxy database and survive version upgrades, a malicious guardrail or callback can persist after the original vulnerability is patched [14]. Upgrading alone does not complete remediation.

The table below summarizes the four LiteLLM entries currently in the KEV catalog, using CISA's catalog data [1]. CISA set an expedited three-day deadline for CVE-2026-42208; the other three entries carry the more common fourteen-day window.

CVE	Weakness (per CISA)	Added to KEV	Federal due date
CVE-2026-42208	SQL injection	2026-05-08	2026-05-11
CVE-2026-42271	Command injection (authenticated, including low-privilege keys)	2026-06-08	2026-06-22
CVE-2026-59822	Improper authentication in MCP Streamable HTTP endpoint	2026-09-02	2026-09-16
CVE-2026-40217	Unprotected alternate channel (custom-code guardrail sandbox escape)	2026-11-18	2026-12-02

Version Confusion Complicates Remediation

The fix information conflicts in ways that could mislead automated scanners and manual triage. The GitHub advisory lists affected versions as `>= 1.81.8, < 1.83.10` and its metadata names 1.83.10 as patched, but its text says the fix is in 1.83.11 and recommends upgrading to "1.83.11 or later" [2].

Obsidian identifies 1.83.14-stable as "the first release that contains the complete fix set" for CVE-2026-47101, CVE-2026-47102, and CVE-2026-40217 [4]. The October 2026 supply chain incident adds a further hazard. `litellm==1.107.0` on PyPI and the Docker tags `v1.107.0`, `latest`, `main-latest`, and `main-stable` served a credential stealer for a few hours on October 21 [10][13]. Ostorlab's current remediation guidance tells readers to pull the `main-stable` tag [8], while other guidance warns that the same tag "tells you nothing about what's inside the container" [9]. The contrast shows how easily a habit of floating tags can conflict with supply chain safety.

Recommendations

Immediate Actions

Organizations running the LiteLLM proxy should first find every instance, including staging environments, developer workstations running the proxy locally, and gateways embedded in agent platforms. They should check the version installed inside each running container, not the version listed in a manifest. Any instance older than 1.83.14-stable should be upgraded to a vetted release at or above that baseline, explicitly excluding the malicious 1.107.0. The release should be pinned by version and image digest, never by a floating tag [4][10][13]. Where an upgrade cannot happen right away, operators should deny `POST /guardrails/test_custom_code` at the reverse proxy or API gateway, as both the vendor and X41 recommend. They should also review whether the production guardrail-creation endpoints need to be reachable from any network outside an administrative enclave [2][3][4].

Because exploitation is confirmed and the payoff is credential theft, teams should hunt for compromise at the same time as they patch. Useful log searches include requests to `/guardrails/test_custom_code` and `/guardrails`, changes to `user_role` through `/user/update` or `/user/bulk_update`, and keys created or updated with broad `allowed_routes` [4]. In the database, administrators should look for unexpected `proxy_admin` users, unfamiliar custom-code guardrails, and new `callbacks` or model `api_base` values that point to unknown hosts [9][14]. Teams should also check provider billing for unusual consumption spikes, because LLMjacking produces sharp spikes in fraudulent charges within hours [5][9]. If any of these signs appear, the host should be treated as compromised. Teams should then rotate the master key, every upstream provider key, the database credentials, all Virtual Keys, and any cloud IAM credentials available to the container, and preserve evidence before rebuilding.

Short-Term Mitigations

Over the following weeks, organizations should reduce the gateway's blast radius whether or not they find compromise. The proxy container should run as a non-root user, as X41 recommends. Management and administrative routes should be reachable only from a restricted network segment, separate from the inference API that applications use [3][9]. The master key should be stored in a secrets manager, rotated on a schedule, and kept out of CI variables and developer configuration. Day-to-day administration should use named, individually accountable admin accounts instead of the shared master key. Virtual Keys should be scoped to the minimum routes and models each consumer needs. Periodic reviews should confirm that no key carries wildcard route access.

Teams should also put detection around the gateway's control plane. Changes to roles, guardrails, callbacks, model endpoints, and key scopes are typically infrequent in steady-state operation, though organizations should baseline their own change rates before setting alert thresholds. Sending these events to a SIEM with alerting turns the gateway's own audit trail into an early warning system. Organizations that adopted LiteLLM's custom-code guardrails should decide whether they need them. Where they are needed, guardrail logic can run in an isolated service called by the gateway, not as code executed inside the gateway process.

Strategic Considerations

The broader lesson is that the policy layer of an AI system is part of its attack surface. Guardrails, content filters, callbacks, and plugin hooks are often added to make AI deployments safer, and they usually need privileged access to every request and response. When these features accept executable code, they become a channel for running that code, and the security of the whole gateway then depends on how that execution is isolated. Security architects should hold AI safety tooling to the same isolation and least-privilege standards as any other code-execution feature, and should prefer declarative policy definitions to arbitrary code where possible.

The KEV history also bears on governance. Four exploited LiteLLM vulnerabilities in one year, plus two supply chain compromises, suggest that LiteLLM, and likely self-hosted AI gateways more broadly, are now a regular target for financially motivated attackers. LLMjacking gives these attackers a direct way to profit [1][5][12][13]. Organizations should cover AI gateways in their vulnerability management SLAs and asset inventories. They should require upgrades to be pinned and verified, and should assess whether a managed or more heavily hardened gateway option matches their risk tolerance. Procurement and third-party risk reviews of AI platforms should ask explicitly how vendors isolate user-supplied code, and how quickly they move fixes for gateway components into their own products.

CSA Resource Alignment

CSA's research note *LiteLLM AI Gateway: KEV-Listed Attack Chain Enables Full Takeover* (June 2026) first documented the three-CVE Obsidian chain that ends in CVE-2026-40217. It recommended v1.83.14-stable as the remediation baseline [12]. The KEV listing confirms that CVE-2026-40217, the final step of that chain, is exploited in the wild. GTIG's account of attackers escalating from Virtual Keys is consistent with, though not confirmed as, use of the full chain the earlier note described, which reinforces that note's advice to treat a low-privilege foothold as a plausible path to compromise of the gateway host. Organizations that followed that guidance in June should now confirm that it was applied to every instance, including ones added since then. CSA's earlier note *LiteLLM AI Gateway: Active Exploitation via MCP Injection* covers CVE-2026-42271, the command injection entry in the KEV table above, and remains relevant to the same patching baseline [16].

LLM Heist: Abusing LiteLLM Callback Hooks Post-Compromise (August 2026) describes what an attacker can do with the administrative access that the CVE-2026-47101/47102 chain provides. That includes traffic redirection, provider-key harvesting, and forged tool calls injected into agent workflows [14]. This is why the present note says that patching CVE-2026-40217 is not enough. Remediation must also confirm that no malicious guardrails, callbacks, or model endpoints were left in the gateway's configuration.

LiteLLM 1.107.0: An Unrevoked Agent Key Becomes a Release Takeover (October 2026) is relevant in two ways [13]. It shows how a single long-lived LiteLLM master key, never revoked, enabled a far larger compromise. That supports the recommendation to rotate the master key and stop using it for routine administration. It also identifies the specific package versions and container tags to avoid during the upgrade this KEV entry requires.

For control mapping, CSA's AI Controls Matrix (AICM) v1.1 gives a framework for putting these recommendations into practice [11]. The Threat and Vulnerability Management (TVM) domain covers bringing AI gateways into patch SLAs and KEV-driven prioritization. The Identity and Access Management (IAM) domain covers master-key handling, role separation, and least-privilege Virtual Key scoping. The Application and Interface Security (AIS) domain covers how custom-code features are isolated. The Logging and Monitoring (LOG) domain covers detection on gateway control-plane changes.

References

[1] Cybersecurity and Infrastructure Security Agency. "Known Exploited Vulnerabilities Catalog." CISA, catalog version 2026.11.18, accessed November 20, 2026.

[2] BerriAI. "LiteLLM has a sandbox escape in custom-code guardrail (CVE-2026-40217, GHSA-wxxx-gvqv-xp7p)." GitHub Advisory Database, May 2026.

[3] X41 D-Sec GmbH. "Advisory X41-2026-001: Guardrail Sandbox Escape in LiteLLM." X41 D-Sec, April 8, 2026.

[4] Obsidian Security. "Breaking LiteLLM: From Low-Privilege User to Admin and RCE (CVE-2026-47101, CVE-2026-47102, CVE-2026-40217)." Obsidian Security, June 11, 2026 (updated September 7, 2026).

[5] Google Threat Intelligence Group and Mandiant. "The Rise of LLMJacking: How Threat Actors Monetize Compromised Cloud and Model Resources." Google Cloud Blog, November 2, 2026.

[6] Google Threat Intelligence Group. "GTIG AI Threat Tracker: Agentic Threat Activity Crosses a Significant Threshold." Google Cloud Blog, November 16, 2026.

[7] Ravie Lakshmanan. "CISA Warns of 4 Exploited Flaws Linked to Ransomware, Cryptomining, and LLMjacking Attacks." The Hacker News, November 19, 2026.

[8] Ostorlab. "LiteLLM Guardrail Sandbox Escape RCE (CVE-2026-40217)." Ostorlab Blog, November 19, 2026.

[9] Alan West. "LiteLLM CVE-2026-40217 Is Exploited From Virtual Keys. How to Check Your Proxy." DEV Community, November 19, 2026.

[10] StepSecurity. "litellm 1.107.0 Compromised on PyPI and Docker: Credential Stealer and Hourly Remote Code Loader." StepSecurity, October 21, 2026.

[11] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

[12] Cloud Security Alliance. "LiteLLM AI Gateway: KEV-Listed Attack Chain Enables Full Takeover." CSA AI Safety Initiative, June 17, 2026.

[13] Cloud Security Alliance. "LiteLLM 1.107.0: An Unrevoked Agent Key Becomes a Release Takeover." CSA AI Safety Initiative, October 26, 2026.

[14] Cloud Security Alliance. "LLM Heist: Abusing LiteLLM Callback Hooks Post-Compromise." CSA AI Safety Initiative, August 5, 2026.

[15] Cloud Security Alliance. "GitLab AI Gateway Sandbox Escape: Template Injection RCE." CSA AI Safety Initiative, October 6, 2026.

[16] Cloud Security Alliance. "LiteLLM AI Gateway: Active Exploitation via MCP Injection." Cloud Security Alliance, 2026.

[17] National Institute of Standards and Technology. "CVE-2026-40217 Detail." National Vulnerability Database, April 10, 2026.

[18] Zope Foundation. "RestrictedPython Documentation." Read the Docs, accessed November 20, 2026.

[19] National Institute of Standards and Technology. "CVE-2023-37271 Detail: RestrictedPython Arbitrary Code Execution via Stack Frame Sandbox Escape." National Vulnerability Database, July 2023.

© 2026 Cloud Security Alliance. AI-assisted Rapid Research.