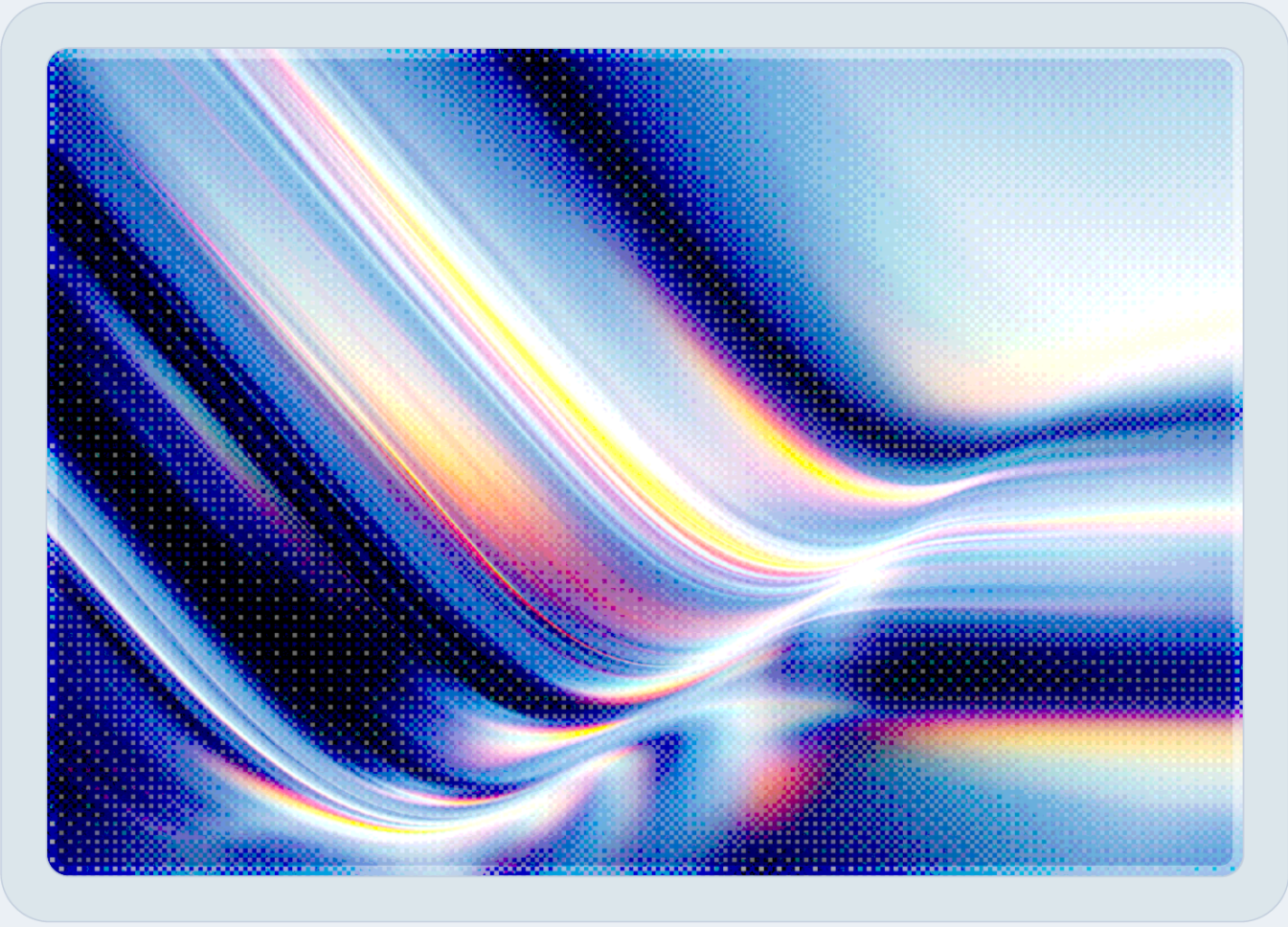
CSAI Foundation | Cloud Security Alliance

LiteLLM Pre-Auth SQL Injection Exploited in 36 Hours

CVE-2026-42208 Targets the LLM Gateway's Stored Provider Credentials

2026-04-28

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

A critical pre-authentication SQL injection in LiteLLM — CVE-2026-42208, CVSS 9.3 — was published as a GitHub Security Advisory on April 20, 2026, indexed in the global GitHub Advisory Database on April 24, and observed under targeted exploitation just 36 hours and seven minutes after that indexing [1][2]. The vulnerability sits in the proxy's Bearer-token verification path: the value of the `Authorization` header is concatenated directly into a SQL query against the `LiteLLM_VerificationToken` table without parameter binding, so any unauthenticated client able to reach the proxy port can break out of the string literal with a single quote and execute arbitrary SQL [3]. All LiteLLM versions from 1.81.16 through 1.83.6 are affected; version 1.83.7 fixes the issue by passing the caller-supplied token as a separate query parameter [3].

What distinguishes this incident from a typical pre-auth SQLi is the nature of the database it exposes. LiteLLM is a widely deployed open-source LLM gateway, used as a unified front end to OpenAI, Anthropic, AWS Bedrock, Google Vertex, and other model providers, and its database aggregates virtual keys, master keys, and the underlying provider credentials in three tables that the attacker targeted directly: `LiteLLM_VerificationToken`, `litellm_credentials`, and `litellm_config` — the exact tables that hold virtual keys, upstream provider credentials, and proxy configuration [1][4]. A successful extraction is therefore closer in blast radius to a multi-cloud account compromise than to a conventional web-application database breach. The incident also follows a March 2026 supply-chain compromise of the LiteLLM PyPI packages, making this the project's second critical security event in roughly five weeks [5].

Organizations operating LiteLLM should act on the following priorities. Upgrade every proxy instance to version 1.83.7 or later as an emergency change rather than a routine maintenance window, given confirmed in-the-wild exploitation [1]. Rotate all LiteLLM-managed virtual keys, the master key, and every upstream provider credential the proxy stores or has stored. Restrict LiteLLM's network exposure so that the proxy is reachable only from authenticated internal callers, never directly from the internet. Audit upstream provider billing and usage records for unfamiliar IP activity beginning at the date the proxy was first deployed in an exposed configuration. Finally, inventory shadow LiteLLM deployments outside the security team's known footprint, because the gateway is frequently stood up by individual developer teams without central registration.

Background

LiteLLM is one of the most widely deployed open-source LLM gateways. Maintained by BerriAI, the project provides a single proxy-and-SDK layer that exposes OpenAI-compatible endpoints in front of more than one hundred underlying model providers, handling key management, rate limiting, cost tracking, retries, and routing [6]. The repository carries more than 45,000 GitHub stars [6], and PyPI download statistics during March 2026 reflected approximately 3.4 million daily installations across direct users and downstream dependents [5][7]. Its appeal is straightforward: the proxy lets a security or platform team issue per-team or per-user "virtual keys" while keeping the high-trust provider credentials centralized, and lets application code call a single endpoint regardless of which underlying model is selected.

That centralization is also what makes LiteLLM a high-value target. A single row in the `litellm_credentials` table can hold an OpenAI organization key with elevated spending limits, an Anthropic console key with workspace-administrator privileges, an AWS Bedrock IAM credential, and an Azure OpenAI deployment secret. A single row in the `LiteLLM_VerificationToken` table represents a virtual key authorized to spend against any of those providers under proxy-defined budgets. The `litellm_config` table holds environment variables and proxy settings, sometimes including additional secrets injected at startup. From an attacker's perspective, a LiteLLM database extraction is a direct path to monetizable cloud and AI-platform credentials with no further exploitation required.

The vulnerability disclosed as CVE-2026-42208 was reported privately to the LiteLLM maintainers by Tencent YunDing Security Lab and published as GHSA-r75f-5x8p-qvmc on April 20, 2026 at 21:14 UTC [3]. The advisory was indexed in the GitHub Advisory Database — making it visible to Dependabot, OSV, and downstream vulnerability scanners — on April 24, 2026 at 16:17 UTC [1]. The first targeted exploitation attempt was recorded by Sysdig's Threat Research Team at 04:24 UTC on April 26, a window of 36 hours and seven minutes after the advisory entered global vulnerability feeds [1]. This timeline is consistent with the rapid weaponization curve now seen across AI development infrastructure, including the recent Marimo notebook RCE that was exploited within ten hours of disclosure [8].

Security Analysis

The Vulnerability: Concatenated Bearer Tokens in Token Lookup

The technical root cause is a CWE-89 SQL injection in a security-critical code path. When a request arrives at any LiteLLM proxy endpoint, the proxy must determine whether the `Authorization: Bearer` value corresponds to a known virtual key. According to the GitHub Security Advisory, the database query used during this proxy API key check "mixed the caller-supplied key value into the query text instead of passing it as a separate parameter" [3]. The Bearer value is interpolated directly into a `SELECT` against the `LiteLLM_VerificationToken` table, so an attacker can terminate the string literal with a single quote and append arbitrary SQL after it. Because the lookup is performed before authentication is decided — by definition, the proxy cannot know whether the caller is authenticated until after this query runs — the injection is fully pre-authentication. Any HTTP client that can reach the proxy port is sufficient to exploit it.

Tencent YunDing's CVSS v4.0 vector for the issue is `AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:H/VA:H`, which scores 9.3 [3]. The "no privileges required" and "no user interaction" components are direct consequences of the pre-auth nature of the bug. The high confidentiality, integrity, and availability impacts reflect that read-write access to the proxy's PostgreSQL backend permits not only credential extraction but also injection of attacker-controlled virtual keys, modification of budget settings, and destruction of audit and usage records. The patched version 1.83.7 rewrites the query so that "the caller-supplied value is now always passed to the database as a separate parameter," restoring parameter-binding at the trust boundary [3].

Targeted Exploitation Within 36 Hours

The Sysdig Threat Research Team observed targeted exploit traffic against LiteLLM honeypots beginning at 04:24 UTC on April 26, 2026 [1]. Several characteristics indicate the activity was targeted rather than opportunistic. Two source IPs — 65.111.27.132 and 65.111.25.67 — operated from adjacent /22 blocks within the same German autonomous system, AS200373 (3xK Tech GmbH), and presented an identical `Python/3.12 aiohttp/3.9.1` user-agent, indicating a single operator rotating addresses rather than a distributed scanner [1]. Across the observation window, the operator sent 29 UNION-based SQL injection payloads, each crafted to extract data from one of three specific tables: the virtual-key store `LiteLLM_VerificationToken`, the upstream-provider credential store `litellm_credentials`, and the proxy configuration store `litellm_config` [1][4].

The payload sequence reveals an operator who arrived with prior schema knowledge rather than a generic SQLi scanner discovering the structure on the fly. Sysdig's analysts noted that after lowercase table references failed, the operator retried with the case-sensitive PascalCase variant `"LiteLLM_VerificationToken"` — a name that matches exactly what appears in LiteLLM's published Prisma schema [1]. The operator also walked the column count systematically, varying the number of `NULL` placeholders from one through six until the underlying query structure was matched. Representative payloads observed in the wild included the following extractions, embedded in the `Authorization` header value:

```
sk-litellm' UNION SELECT api_key,NULL,NULL,NULL,NULL FROM
litellm_verificationtoken--
sk-litellm' UNION SELECT credential_values,NULL,NULL FROM
litellm_credentials--
sk-litellm' UNION SELECT config_value,NULL FROM litellm_config
WHERE param_name='environment_variables'--
```

After 29 unsuccessful UNION variants, the operator deployed a generic `sk-litellm' OR 1=1--` terminal payload, suggesting that the campaign had exhausted its scripted payload list rather than continued adapting [1]. Sysdig reports no confirmed post-exploitation activity using extracted credentials — no observed reuse of stolen keys, no calls to `/key/generate` from the operator's IP space, and no monetization signal — but that absence does not mean credentials were not extracted; it means subsequent abuse, if any, occurred from infrastructure not connected to the original intrusion [1].

A Pattern of LiteLLM Security Incidents

CVE-2026-42208 is not an isolated event in the project's recent history. On March 24, 2026, the LiteLLM maintainers disclosed a suspected supply-chain compromise of two PyPI packages — versions 1.82.7 and 1.82.8 — published during a 40-minute window in which an attacker apparently bypassed the project's CI/CD workflow and uploaded packages directly [5]. Forensic analysis attributed the breach to compromised credentials introduced via the Trivy security scanner used in LiteLLM's build pipeline; the malicious packages contained a credential stealer that exfiltrated environment variables, SSH keys, cloud credentials, Kubernetes tokens, and database passwords to attacker-controlled domains [5][7]. The maintainers responded by removing the packages from PyPI, rotating credentials, engaging Mandiant for incident response, and rebuilding the project's CI/CD pipeline with isolated environments and cosign-signed container images [5].

Read together, these two incidents indicate that LiteLLM occupies an unusual position in the AI infrastructure ecosystem: it is widely deployed enough to be operationally important, sees fast-paced development with frequent releases, and centralizes high-value credentials in a way that makes any compromise high-impact. The maintainers' response to both events has included public disclosure, credential rotation, third-party incident-response engagement, and a rebuilt CI/CD pipeline with isolated environments and cosign-signed images [5]. Even so, the underlying property — that the gateway aggregates the keys to many cloud accounts in a single network-reachable service — is structural and will continue to attract attacker attention. Organizations should plan for additional critical vulnerabilities in this software class, not assume that the most recent patch closes the chapter.

Implications for AI Gateway and Proxy Security

The LLM gateway pattern is increasingly common in production AI deployments. LiteLLM, AI Gateway products from Cloudflare and Kong, the LangChain-adjacent Portkey gateway, and several enterprise vendors all implement the same architectural pattern: a single network-reachable service that holds long-lived, high-value provider credentials and brokers traffic on behalf of upstream applications. CVE-2026-42208 demonstrates that this category is now subject to active exploitation, and that a single defect in the gateway's authentication path is sufficient to convert it into an attacker's credential-extraction tool [15]. The targeting precision documented in this incident — schema-aware payloads aimed at the three exact tables that hold the highest-value secrets — indicates that adversaries are studying the internals of widely deployed AI infrastructure rather than treating it as generic web software. Any organization running an LLM gateway should assume that its data model and authentication implementation are public knowledge, and should harden network exposure, secret storage, and audit logging accordingly.

Recommendations

Immediate Actions

Every organization running LiteLLM must upgrade to version 1.83.7 or later without delay. The vulnerability affects all releases from 1.81.16 through 1.83.6, and the patch closes the injection by passing the Bearer-token value as a parameterized query argument rather than concatenating it into the SQL text [3]. The upgrade should be treated as an emergency change, deployed within hours rather than at the next scheduled maintenance window, given confirmed targeted exploitation against deployed proxies. As an interim workaround for environments that genuinely cannot upgrade in the immediate

term, the LiteLLM maintainers identify setting `disable_error_logs: true` under `general_settings` as a partial mitigation that makes successful exploitation substantially harder, but this is not a substitute for the upgrade [3].

Credential rotation should accompany the upgrade regardless of whether exploitation against a specific proxy has been confirmed. Every virtual key issued by the proxy should be regenerated. The proxy's master key should be rotated. Every upstream provider credential the proxy has stored — OpenAI organization keys, Anthropic console keys, Bedrock IAM credentials, Azure OpenAI deployment secrets, and any others — should be revoked at the provider and reissued. Because the SQL injection grants full read access to `litellm_credentials`, the conservative default assumption is that any credential present in the database during the exposure window is potentially compromised, even where exploitation logs do not show successful extraction.

Organizations should immediately audit network exposure. Any LiteLLM proxy bound to `0.0.0.0` and reachable from the public internet should be moved behind a reverse proxy that enforces authenticated network-layer access, an internal load balancer reachable only from corporate network ranges, or a VPN-gated ingress. The architecture pattern of placing LiteLLM directly on the public internet should be discontinued. Organizations should additionally pull a list of all outbound activity to upstream model providers from the proxy's start date, looking for invocations from unfamiliar IP ranges, unusual model selections, or sudden spend increases that could indicate post-extraction abuse.

Short-Term Mitigations

Detection engineering can supplement the upgrade and rotation work. The `Authorization` header values observed in the Sysdig-documented exploitation attempts contain SQL keywords (`UNION`, `SELECT`, `FROM`, `WHERE`, `--`) and a leading `sk-litellm'` substring with an embedded single quote [1]. Web application firewalls in front of LiteLLM proxies should be configured to block or alert on Bearer header values containing single quotes or SQL syntax, and SIEM rules should fire on any HTTP request to a LiteLLM endpoint where the Authorization header carries SQL keywords. The user-agent `Python/3.12 aiohttp/3.9.1` was used by the targeted operator [1]; while not malicious in itself, it differs from the user-agent strings emitted by LiteLLM's official SDKs and can serve as a weak indicator when correlated with anomalous Authorization values.

The two attacker IPs observed in the documented campaign — 65.111.27.132 and 65.111.25.67, both in AS200373 (3xK Tech GmbH) — should be added to network blocklists and treated as known-bad sources for any AI infrastructure, not solely LiteLLM. Because the operator demonstrated awareness of

LiteLLM's internal schema, additional infrastructure may be in use beyond these two addresses, and similar AS200373 source activity against any AI proxy or gateway should be treated as suspicious by default [1].

Inventory work is also essential. LiteLLM is frequently stood up by individual developer teams without central registration, sometimes inside a Kubernetes cluster, sometimes on a single VM, sometimes as a container in a developer's workstation environment. Security teams should run discovery against their cloud environments for processes listening on the default LiteLLM port (4000), for container images derived from `ghcr.io/berriai/litellm`, and for Kubernetes services labeled with LiteLLM identifiers. Any deployment discovered through this inventory work should be assessed for version, network exposure, and credential storage scope before being permitted to continue operating. Shadow gateways are at elevated risk of running unpatched versions and exposing credentials beyond appropriate scope, since they typically sit outside the security team's patch and credential-management cadence.

Strategic Considerations

The LiteLLM incident reinforces a broader principle: LLM gateways occupy a position structurally analogous to identity providers and secret managers, holding credentials that cascade into many downstream systems on compromise. The category deserves the same rigor organizations apply to identity infrastructure — tightly restricted network access, audit-log review, dedicated incident-response runbooks, and inclusion in the highest patch-priority tier. Organizations should also revisit how upstream provider credentials are issued to their gateways. The common pattern of provisioning a single, broadly scoped provider credential and relying on the gateway to enforce per-user budgets concentrates blast radius. Where providers support it, gateways should be configured with finer-grained, project- or workspace-scoped credentials with per-credential rate and spend caps enforced at the provider.

The speed of weaponization observed here — 36 hours from advisory indexing to targeted, schema-aware exploitation — is consistent with a broader pattern across recent AI infrastructure CVEs, including the Marimo notebook RCE [8]. A 30-day patch SLA — typical of many enterprise vulnerability-management programs — is too slow for AI gateways, notebooks, agent runtimes, and similar tooling, given the disclosure-to-exploitation windows now being observed [1][8]. Organizations should establish a separate AI-infrastructure patch SLA — measured in hours for critical pre-auth issues and days for high-severity authenticated issues — and align their inventory, monitoring, and change-management processes to support it.

CSA Resource Alignment

The CSA MAESTRO framework for agentic AI threat modeling provides the most directly applicable lens for this incident. CVE-2026-42208 sits primarily in MAESTRO Layer 4 (Deployment and Infrastructure), where the LLM gateway runs as a network-reachable runtime, and is governed by Layer 6 (Security and Compliance), the cross-cutting layer that encompasses authentication paths, credential storage, and audit logging [10]. The aggregated provider credentials that the gateway holds also implicate Layer 2 (Data Operations) considerations around how high-value secrets are stored and accessed. MAESTRO's recommended controls — particularly network segmentation isolating AI infrastructure and continuous monitoring of credential-bearing services — would have reduced exposure surface and detection time even where the underlying authentication path itself remained vulnerable.

The CSA AI Controls Matrix (AICM) addresses the governance dimensions across multiple control domains, with the specific domain names drawn from the AICM artifact published at the cited landing page [11]. The Identity and Access Management domain establishes control objectives for properly bounded, parameterized authentication processing — directly addressing the root cause of CVE-2026-42208. The Cryptography, Encryption, and Key Management domain provides controls for secure storage and rotation of credentials held by AI infrastructure components, which would constrain the blast radius of a gateway compromise. The Threat and Vulnerability Management domain provides the scaffolding for incorporating AI-specific software like LiteLLM into the patch-management program with appropriate priority. AICM's role as a superset of the Cloud Controls Matrix is particularly relevant because LLM gateways inherit cloud-infrastructure risk while introducing AI-specific concentration risk that traditional CCM controls do not fully capture.

CSA's Zero Trust guidance speaks directly to the network-exposure dimension [12]. Under Zero Trust principles, no LLM gateway should be reachable from any network that is not itself authenticated and authorized, and the gateway's own authentication must be treated as one layer of defense rather than the sole layer. LiteLLM proxies bound to `0.0.0.0` with no upstream network controls represent direct violations of Zero Trust placement requirements, independent of the software's own authentication. CSA's Securing LLM-Backed Systems guidance and the LLM Threats Taxonomy further address the credential-aggregation pattern that LiteLLM implements, providing the layered controls — authentication, authorization, secret scoping, audit logging, and anomaly detection — that prevent a single proxy defect from cascading into a multi-provider credential compromise [13][14]. The CSA STAR program provides the assurance mechanism: organizations procuring or operating LLM gateways should request STAR-level documentation that specifically addresses the gateway's database-layer security, authentication flow, and credential-handling architecture.

References

[1] Sysdig Threat Research Team. "CVE-2026-42208: Targeted SQL Injection Against LiteLLM's Authentication Path Discovered 36 Hours Following Vulnerability Disclosure." Sysdig, April 2026.

[2] Cyber Kendra. "Hackers Targeted LiteLLM's AI Gateway Just 36 Hours After Critical SQL Injection Flaw Went Public." Cyber Kendra, April 2026.

[3] BerriAI. "GHSA-r75f-5x8p-qvmc: SQL Injection in Proxy API Key Verification." GitHub Security Advisories, April 2026.

[4] CyberSecurityNews. "Critical LiteLLM SQL Injection Vulnerability Exploited in the Wild." CyberSecurityNews, April 2026.

[5] LiteLLM. "Security Update: Suspected Supply Chain Incident." LiteLLM Documentation Blog, March 2026.

[6] BerriAI. "LiteLLM: Call All LLM APIs Using the OpenAI Format." GitHub, 2026.

[7] Snyk. "How a Poisoned Security Scanner Became the Key to Backdooring LiteLLM." Snyk Blog, March 2026.

[8] Sysdig Threat Research Team. "Marimo OSS Python Notebook RCE: From Disclosure to Exploitation in Under 10 Hours." Sysdig, April 2026.

[9] GBHackers. "Critical LiteLLM Flaw Enables Database Attacks Through SQL Injection." GBHackers, April 2026.

[10] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA, February 2025.

[11] Cloud Security Alliance. "AI Controls Matrix." CSA, 2025.

[12] Cloud Security Alliance. "Zero Trust Advancement Center." CSA, 2024.

[13] Cloud Security Alliance. "Securing LLM Backed Systems: Essential Authorization Practices." CSA, 2024.

[14] Cloud Security Alliance. "CSA Large Language Model (LLM) Threats Taxonomy." CSA, 2024.

[15] CyberPress. "Critical LiteLLM SQL Injection Vulnerability Exploited in the Wild." CyberPress, April 2026.

