INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-28 06:32 UTC

Critical SQL Injection Vulnerability in LiteLLM (CVE-2026-42208)

CVE VULNERABILITY | **CRITICAL** | CVSS 9.8 | **CISA KEV**

SCC Item ID	SCC-CVE-2026-0081
Type	CVE Vulnerability
CVE ID	CVE-2026-42208
Severity	CRITICAL
CVSS Base Score	9.8
KEV Status	Yes — CISA Known Exploited Vulnerability
Affected Products	LiteLLM (specific affected versions not confirmed in available sources)
Published	2026-04-27T00:00:00Z
Discovery Source	Vulncheck Kev

Executive Summary

A critical SQL injection vulnerability (CVE-2026-42208) in LiteLLM, an open-source AI gateway framework, allows attackers to execute arbitrary database commands without authentication. The vulnerability is confirmed in the CISA Known Exploited Vulnerabilities catalog, meaning active exploitation is occurring now. Vendor patch availability and affected version ranges are not yet published; treat all deployed LiteLLM instances as potentially vulnerable until vendor confirmation of a safe version is obtained. Organizations using LiteLLM to proxy AI model APIs face immediate risk of data exfiltration, credential theft, and backend database compromise.

Technical Analysis

CVE-2026-42208 is a SQL injection vulnerability (CWE-89) in LiteLLM, an open-source proxy and gateway used to unify access to multiple LLM APIs. CVSS base score is 9.8 (Critical). The vulnerability enables unauthenticated or authenticated attackers to inject arbitrary SQL commands into the backend database via unsanitized input handling. MITRE ATT&CK mapping: T1190 (Exploit Public-Facing Application) and T1059.007 (JavaScript/command injection via scripting). The CVE is confirmed in the CISA Known Exploited Vulnerabilities catalog, confirming active in-the-wild exploitation. Specific affected version ranges and patch identifiers were not retrievable from NVD at analysis time; NVD enrichment and vendor patch details may be pending. CVSS vector string and KEV due date are not yet published. Important note: Source data references three potential CVE identifiers (CVE-2026-42208, CVE-2026-4208, CVE-2026-41208). Confirm the canonical CVE ID against CISA

KEV and NVD directly before actioning, as multiple IDs may indicate duplicate CVE reservations or separate related vulnerabilities. Operators should treat all deployed LiteLLM instances as affected until vendor confirmation of a safe version is obtained.

Action Checklist

1. Step 1: Containment, Immediately restrict external network access to all LiteLLM instances. If LiteLLM is internet-facing, place it behind a WAF or firewall rule that blocks public access until patching is confirmed. Treat all instances as potentially compromised pending investigation.
2. Step 2: Detection, Query application and database logs for anomalous SQL syntax, unusual query lengths, or error spikes originating from LiteLLM service accounts. Look for UNION, SELECT, DROP, INSERT patterns in logged request bodies or query strings. Review database audit logs for unexpected reads or schema queries. Check CISA KEV catalog (https://www.cisa.gov/known-exploited-vulnerabilities-catalog) for any published IOCs associated with this CVE.
3. Step 3: Eradication, Identify the LiteLLM version deployed across all environments. Monitor the LiteLLM GitHub Releases page (https://github.com/BerriAI/litellm/releases) and Security Advisories for a patched version addressing CVE-2026-42208. Apply the patch immediately upon vendor confirmation that the fix resolves this CVE. Until then, enforce input validation or parameterized query controls at the WAF layer if technically feasible.
4. Step 4: Recovery, After patching, rotate all database credentials and API keys accessible to the LiteLLM service. Audit database access logs for signs of exfiltration prior to containment. Verify patch integrity against vendor-published checksums. Resume normal operations only after confirming no unauthorized schema changes or data exports occurred.
5. Step 5: Post-Incident, Review how LiteLLM and similar AI gateway tools are inventoried and monitored in your environment. This vulnerability exposes a gap in AI/ML infrastructure coverage within vulnerability management programs. Add LiteLLM and other open-source AI proxies to your software bill of materials (SBOM) and CVE monitoring pipeline. Update detection rules to cover SQL injection attempts against AI infrastructure endpoints specifically.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and breach notification review if database audit logs confirm unauthorized SELECT queries against tables containing PII, PHI, or stored AI provider API keys, or if LiteLLM's database user had write access and schema changes are detected — both conditions may trigger regulatory notification obligations under GDPR, HIPAA, or state breach notification laws.

Recovery Notes	After patching and credential rotation, monitor PostgreSQL slow query logs and LiteLLM HTTP access logs continuously for 30 days for recurrence of SQLi patterns, as threat actors who successfully exfiltrated credentials may attempt re-entry using harvested API keys or database credentials before rotation is fully propagated. Verify that all AI provider API keys (OpenAI, Anthropic, Azure OpenAI, etc.) stored in LiteLLM's database or config have been revoked at the provider level — not just rotated in LiteLLM's config — since CVE-2026-42208 allows unauthenticated database reads and all stored secrets must be treated as fully compromised. Conduct a configuration review of LiteLLM's deployment to confirm it is no longer directly internet-exposed and that the principle of least privilege is enforced on its database service account, restricting it to only the schemas and operations required for normal operation.
Forensic Artifacts	LiteLLM HTTP access logs (default location: journald or /var/log/litellm/) covering 30 days pre-detection — unauthenticated SQLi against CVE-2026-42208 will appear as anomalous request bodies or query strings containing SQL metacharacters (quotes, semicolons, UNION keywords) in pre-authentication endpoints | PostgreSQL pg_log query logs for the LiteLLM database service account — successful exploitation produces information_schema and pg_catalog enumeration queries followed by bulk SELECT statements; failed probes generate PG::SyntaxError entries that form a reconnaissance timeline | PostgreSQL WAL (Write-Ahead Log) segment archives from /var/lib/pgsql/data/pg_wal/ — provides a forensically complete, append-only record of every transaction executed against the LiteLLM database during the exposure window, including any attacker-issued INSERT, UPDATE, or exfiltration-oriented SELECT operations | LiteLLM configuration files (config.yaml, .env, or environment variables captured from /proc/[pid]/environ) — post-exploitation persistence may manifest as attacker-added model routing entries, rogue API keys, or modified database connection strings inserted via the compromised DB write access | Network pcap of traffic to LiteLLM's listening port (default 4000/tcp) captured before isolation — preserves raw exploit payloads including SQL injection strings, enabling extraction of attacker-controlled SQL commands and identification of the specific injection point (parameter name, endpoint path) for detection rule authoring

Per-Action IR Details

Step 1: Containment — Immediately restrict external network access to all LiteLLM instances. If LiteLLM is internet-facing, place it behind a WAF or firewall rule that blocks public access until patching is confirmed. Treat all instances as compromised pending investigation.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices)

Compensating: On Linux hosts running LiteLLM, immediately execute: `iptables -I INPUT -p tcp --dport 4000 -j DROP` (LiteLLM default port) to block inbound traffic without a WAF. On Windows, use: `netsh advfirewall firewall add rule name='Block LiteLLM' dir=in action=block protocol=TCP localport=4000`. If LiteLLM sits behind nginx or another reverse proxy, add `deny all;` to the upstream block in the nginx config and reload. Document the exact timestamp of isolation for the incident timeline.

Evidence: Before isolating, capture a full packet capture of active connections to LiteLLM's listening port using `tcpdump -i any -w litellm_preisolation_$(date +%F_%T).pcap port 4000` — this preserves in-flight exploit payloads. Record all established TCP connections via `ss -tnp | grep 4000` or `netstat -anp | grep 4000` and save to file. Dump the LiteLLM process memory if feasible using `gcore $(pgrep -f litellm)` before killing or isolating the service, as SQL injection exploit strings or injected payloads may reside in heap memory.

Step 2: Detection — Query application and database logs for anomalous SQL syntax, unusual query lengths, or error spikes originating from LiteLLM service accounts. Look for UNION, SELECT, DROP, INSERT patterns in logged request bodies or query strings. Review database audit logs for unexpected reads or schema queries. Check CISA KEV catalog (https://www.cisa.gov/known-exploited-vulnerabilities-catalog) for any published IOCs associated with this CVE.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST SI-4 (System Monitoring), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: If no SIEM is available, run the following directly against LiteLLM application logs (typically at `/var/log/litellm/` or journald): `grep -iE "(UNION|SELECT\s+\*|DROP\s+TABLE|INSERT\s+INTO|1=1|OR\s+1|--\s|;\s*SELECT)" /var/log/litellm/*.log > sqli_hits.txt`. For PostgreSQL (LiteLLM's default backend DB), enable query logging by setting `log_min_duration_statement = 0` in `postgresql.conf` and then grep `pg_log/` for queries issued by the LiteLLM DB user: `grep -i 'litellm_user\|ERROR\|syntax error' /var/lib/pgsql/data/log/postgresql-*.log`. For error spike detection, run: `awk '/500|SQL|syntax/{count++} END{print count}' /var/log/litellm/access.log` and compare against a 7-day rolling baseline.

Evidence: Preserve LiteLLM HTTP access logs (full request URI and body if verbose logging is enabled) from the period 30 days prior to detection — CVE-2026-42208 is an unauthenticated injection, so exploit attempts appear in pre-auth request logs. Capture PostgreSQL or SQLite `pg_log` query logs for the LiteLLM service account, specifically filtering for `information_schema`, `pg_catalog`, or `sys.tables` queries which indicate schema enumeration. Export database error logs showing `PG::SyntaxError` or `SQLite3::Exception` entries, as failed injection probes generate parse errors before a successful payload is crafted.

Step 3: Eradication — Identify the LiteLLM version deployed across all environments. Monitor the LiteLLM GitHub repository (https://github.com/BerriAI/litellm) and the vendor's official release notes for a patched version addressing CVE-2026-42208. Apply the patch immediately upon release. Until then, enforce input validation or parameterized query controls at the WAF layer if technically feasible.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-10 (Information Input Validation), NIST CM-6 (Configuration Settings), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Determine installed LiteLLM version via `pip show litellm | grep Version` or `pip3 show litellm`. If a patch is not yet released, implement WAF rules using ModSecurity (free, open-source) with the OWASP Core Rule Set (CRS) SQL injection ruleset — specifically enable rules 942100–942500 which target UNION-based, error-based, and boolean-based SQLi. Alternatively, deploy a Nginx `ngx_http_rewrite_module` block that rejects requests containing raw SQL metacharacters: `if ($request_body ~* "(union|select|insert|drop|;|--)" ) { return 403; }`. Document the LiteLLM version and deployment location for every environment in a simple CSV asset register as a prerequisite to tracking patch status.

Evidence: Before patching, preserve the vulnerable LiteLLM package artifact: `pip download litellm==[current_version] -d ./litellm_vuln_backup/` — this retains the vulnerable binary for post-incident forensic diffing against the patched release. Capture the current database schema via `pg_dump --schema-only -U litellm_user litellm_db > schema_prePatch_$(date +%F).sql` to establish a baseline for detecting any unauthorized schema modifications made during exploitation. Record all running LiteLLM process arguments via `ps aux | grep litellm` and environment variables via `cat /proc/$(pgrep -f litellm)/environ | tr '\0' '\n'` to identify any injected environment-level persistence.

Step 4: Recovery — After patching, rotate all database credentials and API keys accessible to the LiteLLM service. Audit database access logs for signs of exfiltration prior to containment. Verify patch integrity against vendor-published checksums. Resume normal operations only after confirming no unauthorized

schema changes or data exports occurred.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST IA-5 (Authenticator Management), NIST AU-11 (Audit Record Retention), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 5.2 (Use Unique Passwords), CIS 3.4 (Enforce Data Retention)

Compensating: Rotate LiteLLM's database credentials immediately by running `ALTER USER litellm_user WITH PASSWORD 'new_strong_password';` in psql, then update the `DATABASE_URL` environment variable or LiteLLM config file and restart the service. Rotate all AI provider API keys (OpenAI, Anthropic, etc.) stored in LiteLLM's config or database by revoking and reissuing them via each provider's dashboard — CVE-2026-42208 allows unauthenticated DB reads, meaning these keys are presumed exfiltrated. Verify patch integrity by comparing `sha256sum $(pip show litellm | grep Location | awk '{print $2}')/litellm-*.dist-info/RECORD` against the checksum published in the LiteLLM GitHub release notes for the patched version. Use `pgaudit` (free PostgreSQL extension) to generate a retroactive query audit showing all SELECT statements executed by the LiteLLM service account over the exposure window.

Evidence: Before resuming operations, export and preserve all database transaction logs covering the exposure window: in PostgreSQL, archive WAL (Write-Ahead Log) segments from `/var/lib/pgsql/data/pg_wal/` — these contain a complete record of every query executed, including any exfiltration-oriented bulk SELECT operations. Diff the current database schema against the pre-patch baseline captured in Step 3: `diff schema_prePatch.sql <(pg_dump --schema-only -U litellm_user litellm_db)` to identify any tables, stored procedures, or triggers added by an attacker. Check the LiteLLM configuration file (typically `config.yaml` or `.env`) for any newly added model endpoints or API keys that were not present before the incident, indicating attacker-inserted persistence via the compromised DB write access.

Step 5: Post-Incident — Review how LiteLLM and similar AI gateway tools are inventoried and monitored in your environment. This vulnerability exposes a gap in AI/ML infrastructure coverage within vulnerability management programs. Add LiteLLM and other open-source AI proxies to your software bill of materials (SBOM) and CVE monitoring pipeline. Update detection rules to cover SQL injection attempts against AI infrastructure endpoints specifically.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity

Controls: NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SI-5 (Security Alerts, Advisories, and Directives), NIST RA-3 (Risk Assessment), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Generate an SBOM for AI/ML infrastructure using `syft` (free, Anchore): `syft packages dir:/opt/litellm -o cyclonedx-json > litellm_sbom.json` — this produces a CycloneDX SBOM that can be fed into `grype` (free) for ongoing CVE scanning: `grype sbom:litellm_sbom.json`. Write a Sigma detection rule targeting SQL injection payloads in HTTP request bodies destined for LiteLLM's API port (default 4000/tcp) and publish it to your log pipeline (even if that pipeline is just a cron job running grep against log files). Subscribe to LiteLLM's GitHub repository security advisories via the 'Watch → Security alerts' feature to receive future CVE notifications without a commercial feed. Add `litellm` as a monitored package in OSV-Scanner (free, Google): `osv-scanner --lockfile requirements.txt` if LiteLLM is managed via pip requirements.

Evidence: Compile the full incident timeline from collected artifacts — HTTP logs, DB query logs, WAL archives, and pcap files — and document the earliest evidence of exploitation attempts versus confirmed exploitation, to establish the true exposure window for breach notification assessment. Retain all forensic artifacts for a minimum of 12 months per NIST AU-11 (Audit Record Retention) recommendations, as regulatory inquiries related to AI API credential exposure (OpenAI keys, Anthropic keys, etc.) may arise weeks after the incident closes. Document which AI provider API keys were confirmed or presumed exfiltrated, as these may have been used to make fraudulent API calls chargeable to the organization's account — check each provider's usage dashboard for anomalous call volume during the exposure window.

Detection Guidance

Monitor application logs for SQL metacharacters (single quotes, double dashes, semicolons, UNION/SELECT keywords) in request parameters routed through LiteLLM. Enable database-level query logging and alert on: queries sourced from the LiteLLM service account that exceed normal length thresholds; error responses containing SQL syntax fragments; unusually high query rates or bulk SELECT operations. At the network layer, inspect HTTP request bodies to LiteLLM API endpoints for encoded or obfuscated SQL payloads. If a WAF is in place, review blocked request logs for SQL injection signatures targeting LiteLLM routes. No confirmed public IOCs (IPs, domains, hashes) were available in source data at analysis time; check CISA KEV (https://www.cisa.gov/known-exploited-vulnerabilities-catalog) for any published indicators.

Framework Mappings

MITRE-ATTACK

- **T1190** — Exploit Public-Facing Application
- **T1059.007** — JavaScript

NIST-800-53R5

- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **SI-10** — Information Input Validation
- **SR-2** — Supply Chain Risk Management Plan
- **IR-5** — Incident Monitoring

OWASP-TOP10-2021

- **A03:2021** — Injection

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **15.1** — Establish and Maintain an Inventory of Service Providers

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.AE-08** — Incidents are declared when adverse events meet the defined incident criteria

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1190	Exploit Public-Facing Application	Initial-Access
T1059.007	JavaScript	Execution

Sources

Source	URL	Tier
vulncheck_kev	https://nvd.nist.gov/vuln/detail/CVE-2026-42208	T1
CVE-2026-4208 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-4208	T1
LiteLLM Contains Critical SQL Injection Vulnerability	https://letsdatascience.com/news/litellm-contains-critical-sql-inje...	T3
CVE-2026-41208 - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-41208	T1
Critical LiteLLM Flaw Enables Database Attacks Through SQL ...	https://gbhackers.com/critical-litellm-flaw-enables-database-attacks/	T3
CISA KEV	https://www.cisa.gov/known-exploited-vulnerabilities-catalog	T1

Generated 2026-04-28 06:32 UTC by TJS Security Command Center