INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-04 13:37 UTC

Mercor AI Startup Hit by LiteLLM Open-Source Supply Chain Attack

THREAT CAMPAIGN | **HIGH** | CVSS 8.1

SCC Item ID	SCC-CAM-2026-0146
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	8.1
Affected Products	Mercor (AI startup, production environment); LiteLLM (open-source AI LLM proxy tool, specific version unconfirmed at analysis time)
Published	2026-04-02
Discovery Source	Gemini

Executive Summary

AI hiring startup Mercor suffered a security incident after threat actor group TeamPCP compromised LiteLLM, an open-source tool Mercor used in its production AI pipeline. The attack exploited a supply chain dependency, giving attackers a pathway into Mercor's environment without directly targeting Mercor's own systems. Meta has suspended active projects with Mercor following disclosure, indicating business-level impact from the supply chain compromise and illustrating how upstream open-source compromises can trigger downstream business consequences at scale.

Technical Analysis

Attack vector: supply chain compromise of LiteLLM, an open-source Python-based unified API proxy for large language model providers (PyPI package: litellm). Threat actor TeamPCP is attributed to the upstream project compromise. The attack chain maps to MITRE ATT&CK T1195 (Supply Chain Compromise), T1195.001 (Compromise Software Dependencies and Development Tools), T1072 (Software Deployment Tools), and T1059 (Command and Scripting Interpreter). Relevant CWEs: CWE-494 (Download of Code Without Integrity Check), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-1357 (Reliance on Insufficiently Trustworthy Component). No CVE has been assigned to the LiteLLM compromise. Specific affected LiteLLM version range, malicious commit hash, and payload type have not been confirmed from primary sources and are not represented here. Patch status: unconfirmed, verify directly against the official LiteLLM GitHub repository (https://github.com/BerriAI/litellm) and PyPI release history. No vendor CVSS vector published. No CISA KEV listing at time of analysis.

Action Checklist

1. Step 1: Containment, Immediately identify all systems consuming LiteLLM from PyPI or direct GitHub installs. Isolate any environment where LiteLLM is installed in a production or AI pipeline context until version integrity is confirmed. If Mercor services or APIs are integrated into your environment, treat those connections as potentially compromised pending further disclosure.
2. Step 2: Detection, Audit installed LiteLLM versions across all environments: run 'pip show litellm' or query your software inventory/SBOM for the litellm package. Review dependency installation logs, CI/CD pipeline logs, and package manager audit trails for unexpected version changes or installs during the compromise window. Monitor for anomalous outbound network connections from hosts running LiteLLM, unexpected process execution spawned by Python interpreters, and any new scheduled tasks or persistence mechanisms on affected hosts. No confirmed IOC hashes or C2 infrastructure are available from primary sources at this time.
3. Step 3: Eradication, Cross-reference your installed LiteLLM version against the official LiteLLM GitHub commit history and PyPI release checksums to identify whether your version falls within any announced compromise window. Remove and reinstall LiteLLM only from a confirmed clean release once the LiteLLM maintainers publish remediation guidance. Enforce package integrity verification (pip hash checking or equivalent) before reinstallation. Rotate any credentials, API keys, or tokens accessible to processes running LiteLLM.
4. Step 4: Recovery, After reinstalling a verified clean version, validate that no unauthorized changes persist in your environment: re-scan hosts for malware, verify no new user accounts or persistence mechanisms were introduced, and confirm API key rotation is complete. Re-enable LiteLLM-dependent services only after integrity checks pass. Increase monitoring on AI pipeline components for 30 days post-remediation.
5. Step 5: Post-Incident, Conduct a dependency audit across all AI/ML toolchain components to identify other open-source packages consumed without integrity verification. Implement or enforce SBOM generation for all production deployments. Require cryptographic hash verification for all PyPI and GitHub-sourced packages in CI/CD pipelines. Map open-source AI toolchain dependencies to your risk register. Evaluate vendor risk for third-party AI service integrations (e.g., review data-sharing scope with AI training partners). This incident exposed CWE-494 and CWE-829 class gaps; prioritize controls around dependency pinning, supply chain integrity verification, and least-privilege access for pipeline components.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to CISO and legal counsel if forensic analysis of LiteLLM process network logs reveals confirmed exfiltration of PII processed through the AI hiring pipeline (candidate data, résumés, assessment results), if any Mercor API integration transmitted data to external endpoints during the compromise window, or if the team lacks capacity to perform package forensics and credential rotation within 4 hours of detection.

Recovery Notes	Reinstall LiteLLM only after official remediation guidance is published by the LiteLLM maintainers on their GitHub Security Advisories page; do not rely on version number alone, as supply chain attacks may affect multiple version tags. Verify clean reinstall by comparing SHA256 of all files in the litellm site-packages directory against the PyPI-published RECORD file for the target version. Maintain elevated monitoring on all AI pipeline components — including model inference logs, API gateway access logs, and outbound network connections from Python interpreter processes — for a minimum of 30 days, given the TeamPCP group's supply chain methodology may include delayed-activation payloads.
Forensic Artifacts	LiteLLM site-packages directory file hashes: SHA256 of every .py file in /litellm/ compared against PyPI published RECORD file — a supply chain backdoor inserted by TeamPCP would manifest as a hash mismatch in __init__.py, utils.py, or proxy-related modules that handle API routing. | Python interpreter process network connections: Active and historical outbound TCP connections from the python/python3 process running LiteLLM, captured via /proc//net/tcp on Linux or ETW network events on Windows — TeamPCP's implant would likely beacon to a C2 over HTTPS (port 443) from the LiteLLM proxy process handling AI model API calls. | CI/CD pipeline dependency resolution logs: GitHub Actions runner logs, Jenkins console output, or GitLab CI job traces showing the exact pip install command, resolved LiteLLM version, and wheel hash at the time of the supply chain compromise — the delta between expected and actual hash is the primary indicator for TeamPCP's PyPI tampering. | AI pipeline API call logs: LLM provider access logs (OpenAI usage logs, Anthropic API logs, or equivalent) showing all model inference requests proxied through LiteLLM during the compromise window — anomalous prompt content, unexpected model endpoints, or data exfiltration disguised as model input would appear here. | Environment variable exposure snapshot: Contents of the process environment (/proc//environ on Linux) for the LiteLLM service process at time of discovery — LiteLLM in production AI pipelines typically holds LLM provider API keys, database credentials, and third-party service tokens in environment variables, all of which are in scope for credential rotation and exposure assessment.

Per-Action IR Details

Step 1: Containment — Immediately identify all systems consuming LiteLLM from PyPI or direct GitHub installs. Isolate any environment where LiteLLM is installed in a production or AI pipeline context until version integrity is confirmed. If Mercor services or APIs are integrated into your environment, treat those connections as potentially compromised pending further disclosure.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run 'pip show litellm' and 'pip list --path ' across all hosts; use osquery with 'SELECT name, version, path FROM python_packages WHERE name = "litellm";' to enumerate installs fleet-wide. Block outbound connections from affected Python interpreter processes using host-based firewall rules (iptables -A OUTPUT -p tcp --dport 443 -m owner --uid-owner -j DROP) until integrity is confirmed. For GitHub-sourced installs, check 'git log --oneline' in the cloned repo directory for unexpected commits post-installation.

Evidence: Before isolating, capture: (1) full 'pip list' output and 'pip show litellm' including Location field to identify install path; (2) netstat/ss output showing active connections from the Python process running LiteLLM; (3) process tree snapshot (ps auxf on Linux, 'Get-Process | Select-Object Id,Name,Path,CommandLine' on Windows) to identify what spawned the LiteLLM process; (4) contents of the litellm package directory including any .py files and __init__.py for hash comparison against known-good PyPI release.

Step 2: Detection — Audit installed LiteLLM versions across all environments: run 'pip show litellm' or query your software inventory/SBOM for the litellm package. Review dependency installation logs, CI/CD pipeline logs, and package manager audit trails for unexpected version changes or installs during the compromise window. Monitor for anomalous outbound network connections from hosts running LiteLLM, unexpected process execution spawned by Python interpreters, and any new scheduled tasks or persistence mechanisms on affected hosts. No confirmed IOC hashes or C2 infrastructure are available from primary sources at this time.

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 SI-5 (Security Alerts, Advisories, and Directives), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 8.2 (Collect Audit Logs)

Compensating: Query CI/CD pipeline logs (GitHub Actions logs, Jenkins build console output, GitLab CI job traces) for any 'pip install litellm' commands that resolved to an unexpected version or hash. Use Sysmon Event ID 1 (Process Creation) filtered on ParentImage containing 'python' or 'pip' to detect unusual child processes spawned from the LiteLLM runtime. On Linux, audit /var/log/auth.log and /var/log/syslog for cron job additions or systemd unit file changes. Deploy the Sigma rule for 'Suspicious Python Script Execution' (github.com/SigmaHQ/sigma, rule category process_creation) tuned to flag python.exe or python3 spawning network tools (curl, wget, nc). Use Wireshark or tcpdump to capture 60 minutes of traffic from the affected host: 'tcpdump -i eth0 -w litellm_capture.pcap host ' for later analysis.

Evidence: Capture before analysis concludes: (1) pip installation logs at ~/.pip/pip.log or %APPDATA%\pip\pip.log showing the exact version resolved and timestamp; (2) CI/CD pipeline artifact logs showing the requirements.txt or pyproject.toml dependency resolution for litellm during the suspected compromise window; (3) Python interpreter access logs — on Linux check ~/.python_history and audit /proc//cmdline for running Python processes; (4) network flow data (NetFlow, pcap, or VPC Flow Logs if cloud-hosted) showing outbound connections from the LiteLLM service process, particularly to non-whitelisted external IPs on ports 443 or 80; (5) GitHub Actions or equivalent CI runner logs for the litellm dependency installation step, specifically the resolved hash vs. expected hash.

Step 3: Eradication — Cross-reference your installed LiteLLM version against the official LiteLLM GitHub commit history and PyPI release checksums to identify whether your version falls within any announced compromise window. Remove and reinstall LiteLLM only from a confirmed clean release once the LiteLLM maintainers publish remediation guidance. Enforce package integrity verification (pip hash checking or equivalent) before reinstallation. Rotate any credentials, API keys, or tokens accessible to processes running LiteLLM.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST IA-5 (Authenticator Management) — for credential rotation, CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Verify PyPI release integrity by running 'pip download litellm== --no-deps -d /tmp/litellm_verify' and comparing SHA256 of the downloaded .whl against the hash published on pypi.org/project/litellm//#files. Use 'pip install --require-hashes -r requirements.txt' with explicit hash pins to enforce integrity on reinstall. For credential rotation, enumerate all environment variables and secrets manager entries accessible to the LiteLLM service account using 'printenv | grep -iE "key|token|secret|api"' and rotate every identified credential through your secrets manager (AWS Secrets Manager, HashiCorp Vault, or equivalent). Use YARA rules targeting known supply chain backdoor patterns (file write to site-packages, unexpected imports of socket or subprocess in package __init__.py) to scan the Python environment post-removal: 'yara -r supply_chain.yar /usr/lib/python3/'.

Evidence: Before removing the compromised package: (1) copy the entire litellm package directory from site-packages to an isolated forensic volume — 'cp -r $(pip show litellm | grep Location | cut -d" " -f2)/litellm /forensics/litellm_evidence/'; (2) record SHA256 of every .py file in that directory: 'find /forensics/litellm_evidence/

-name "*.py" -exec sha256sum {} \;'; (3) export all environment variables visible to the LiteLLM process before rotation to document what credentials may have been exposed; (4) collect any LLM API call logs (OpenAI, Anthropic, or model provider logs) accessible to LiteLLM during the compromise window to assess data exfiltration scope; (5) snapshot the PyPI installation metadata file at /litellm-.dist-info/RECORD for comparison against the known-good PyPI RECORD file.

Step 4: Recovery — After reinstalling a verified clean version, validate that no unauthorized changes persist in your environment: re-scan hosts for malware, verify no new user accounts or persistence mechanisms were introduced, and confirm API key rotation is complete. Re-enable LiteLLM-dependent services only after integrity checks pass. Increase monitoring on AI pipeline components for 30 days post-remediation.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-3 (Malicious Code Protection), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 5.1 (Establish and Maintain an Inventory of Accounts)

Compensating: Run ClamAV full scan on affected hosts post-reinstall: 'clamscan -r --infected --remove /home /opt /var /tmp'. Enumerate all local and service accounts created or modified during the compromise window using 'getent passwd | awk -F: "$3 >= 1000"' on Linux or 'Get-LocalUser | Where-Object {$_.LastLogon -gt (Get-Date).AddDays(-30)}' on Windows. Check for new cron jobs: 'for user in $(cut -f1 -d: /etc/passwd); do crontab -u $user -l 2>/dev/null; done'. Verify Python site-packages integrity by recomputing SHA256 hashes of all installed packages and comparing against PyPI published hashes using 'pip-audit' (free tool). Confirm API key rotation by testing that old keys return 401 from each affected service endpoint.

Evidence: Before re-enabling services: (1) re-run the full osquery python_packages query to confirm only the verified clean LiteLLM version is present; (2) collect /etc/passwd, /etc/shadow (hashes only), and /etc/cron.d/* snapshots to establish a clean baseline for comparison against the pre-incident state; (3) review AI pipeline service logs (model inference logs, API gateway access logs) for the 30 days preceding the incident to identify any anomalous LLM API calls that may indicate data extraction through the compromised LiteLLM proxy; (4) capture a network baseline of expected outbound connections from the AI pipeline host post-remediation using 'ss -tunap' and document it for anomaly detection over the 30-day monitoring period.

Step 5: Post-Incident — Conduct a dependency audit across all AI/ML toolchain components to identify other open-source packages consumed without integrity verification. Implement or enforce SBOM generation for all production deployments. Require cryptographic hash verification for all PyPI and GitHub-sourced packages in CI/CD pipelines. Map open-source AI toolchain dependencies to your risk register. Evaluate vendor risk for third-party AI service integrations (e.g., review data-sharing scope with AI training partners). This incident exposed CWE-494 and CWE-829 class gaps — prioritize controls around dependency pinning, supply chain integrity verification, and least-privilege access for pipeline components.

NIST Phase: Post Incident

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

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST RA-3 (Risk Assessment), NIST SA-12 (Supply Chain Protection), NIST IR-8 (Incident Response Plan), 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 all production Python environments using 'pip-audit --format=cyclonedx' or 'syft dir:/opt/app -o cyclonedx-json' (Syft is free, open-source). Enforce pip hash pinning in all CI/CD pipelines by adding '--require-hashes' to pip install commands and pinning all dependencies in requirements.txt using 'pip-compile --generate-hashes' from pip-tools. For GitHub-sourced dependencies, require commit SHA pinning in requirements files rather than branch references. Add 'pip-audit' as a mandatory CI/CD gate step that fails the build on any known-vulnerable package. Document all open-source AI/ML packages (LiteLLM, LangChain, Hugging Face Transformers, etc.) in a risk register entry with owner, version, last audit date, and integrity verification method.

Evidence: For the lessons-learned record: (1) the full SBOM snapshot from all production Python environments at the time of the incident, generated retroactively if not previously maintained; (2) CI/CD pipeline configuration files (requirements.txt, pyproject.toml, Pipfile.lock) from the compromised deployment showing whether hash pinning was absent; (3) the LiteLLM package RECORD file diff between the compromised and clean versions to document exactly which files were tampered; (4) any AI training data pipeline logs showing what data was processed through LiteLLM during the compromise window, relevant to assessing exposure of Mercor's AI hiring workflow data or Meta-connected training datasets; (5) vendor risk documentation for Mercor's API integration, including data-sharing agreements and the scope of data accessible to the Mercor-integrated services prior to Meta's suspension of the partnership.

Detection Guidance

No confirmed IOCs (hashes, IPs, domains) have been released from primary sources at time of analysis; do not use unverified indicators. Focus detection on behavioral and inventory signals: (1) SBOM / package inventory: query for litellm package presence and version across all hosts and containers, flag any version installed during the suspected compromise window once that window is published by LiteLLM maintainers. (2) Process telemetry: alert on Python interpreter processes spawning unexpected child processes (e.g., cmd.exe, bash, curl, wget) on hosts running LiteLLM. (3) Network telemetry: baseline and monitor outbound connections from LiteLLM-adjacent processes; flag connections to new or uncategorized external endpoints. (4) CI/CD logs: review pipeline execution logs for dependency resolution events, unexpected package version pulls, or failed hash checks. (5) Credential exposure: audit whether API keys for LLM providers (OpenAI, Anthropic, etc.) configured in LiteLLM were accessible to the compromised component and treat those keys as potentially exposed. Monitor for anomalous API usage against those provider accounts. Update detection rules as primary-source IOCs are released by LiteLLM maintainers or CISA.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://github.com/BerriAI/litellm`	Official LiteLLM repository — monitor for maintainer incident disclosure, affected version range, and remediation commits. Not a malicious IOC.	HIGH

Framework Mappings

MITRE-ATTACK

- **T1072** — Software Deployment Tools
- **T1195** — Supply Chain Compromise
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059** — Command and Scripting Interpreter

NIST-800-53R5

- **SA-9** — External System Services
- **SR-2** — Supply Chain Risk Management Plan
- **SR-3** — Supply Chain Controls and Processes

- **SI-7** — Software, Firmware, and Information Integrity
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CM-3** — Configuration Change Control

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **15.1** — Establish and Maintain an Inventory of Service Providers

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

ISO-27001-2022

- **A.5.21** — Managing information security in the ICT supply chain

SOC2-TSC

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1072	Software Deployment Tools	Execution
T1195	Supply Chain Compromise	Initial-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059	Command and Scripting Interpreter	Execution

Sources

Source	URL	Tier
Mercor says it was hit by cyberattack tied to compromise of open ...	https://techcrunch.com/2026/03/31/mercor-says-it-was-hit-by-cyberat...	T2

Source	URL	Tier
Twin cybersecurity incidents leave AI industry shaken - Yahoo Finance	https://finance.yahoo.com/sectors/technology/article/twin-cybersecu...	T3
Mercor confirms security incident tied to LiteLLM supply chain attack	https://therecord.media/mercor-confirms-security-incident-tied-to-l...	T3
A $10 billion AI startup just got gutted because a security scanner ...	https://x.com/aakashgupta/status/2039257968265540061	T3
Mercor Hit by LiteLLM Supply Chain Attack - SecurityWeek	https://www.securityweek.com/mercor-hit-by-litellm-supply-chain-att...	T3

Generated 2026-04-04 13:37 UTC by TJS Security Command Center