LiteLLM PyPI Backdoor: Credential Theft in AI Toolchains

TeamPCP's Poisoned Package Exposed Systemic Credential Risk Across AI Infrastructure

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-27

Key Takeaways

- On March 24, 2026, threat actor TeamPCP published backdoored versions of LiteLLM (v1.82.7 and v1.82.8) to PyPI, embedding credential-stealing malware in one of the AI developer ecosystem's most widely used packages, with approximately 95 million monthly downloads [1][4].
- The LiteLLM compromise was the fourth wave of a coordinated TeamPCP supply chain campaign. The attacker obtained LiteLLM's PyPI publish token from credentials previously harvested when LiteLLM's CI/CD pipeline ran a backdoored version of the Trivy security scanner five days earlier [2][3].
- The v1.82.8 payload introduced a `.pth` file (`litellm_init.pth`) that executes the credential stealer on every Python interpreter startup—not only when LiteLLM is invoked—meaning any Python process on an affected host, including unrelated applications and cron jobs, would trigger exfiltration [1][2].
- The credential scope is uniquely dangerous for AI infrastructure: beyond cloud provider credentials, SSH keys, and Kubernetes secrets, the stealer targets `.env` files where AI-using organizations routinely store LLM API keys for OpenAI, Anthropic, Google Gemini, and similar services.
- PYSEC-2026-2 covers the malicious packages. The last confirmed clean version is 1.82.6; organizations that installed v1.82.7 or v1.82.8 must treat the host as fully compromised and rotate all credentials, including LLM API keys, immediately [1][5].
- Three days after the LiteLLM incident, TeamPCP compromised the Telnyx Python library on PyPI (versions 4.87.1 and 4.87.2), demonstrating the threat actor's continued activity using credentials harvested from prior waves [14]. Defenders should not assume the campaign is resolved.

Background

LiteLLM's Role in the AI Developer Ecosystem

LiteLLM is a Python library that provides a unified interface to more than 100 large language model providers, including OpenAI, Anthropic, Google Gemini, AWS Bedrock, Azure OpenAI, Cohere, and others. Rather than integrating each provider's SDK individually, development teams use LiteLLM as a single abstraction layer to call any LLM API using a standardized format. This architectural role makes LiteLLM ubiquitous in AI-native applications: it appears in agentic AI frameworks, LLM-powered microservices, internal developer tooling, and production inference pipelines at organizations ranging from early-stage startups to large enterprises [1].

The package's download volume reflects this centrality. At approximately 95 million monthly PyPI downloads as of March 2026 [1][4], LiteLLM is among the highest-volume AI infrastructure packages in the Python ecosystem. It is also embedded in many higher-level frameworks as an undeclared transitive dependency, meaning organizations may run LiteLLM without having explicitly installed it. This breadth of deployment makes LiteLLM a high-value target for supply chain attacks: a single compromised release can reach a vast number of AI production environments simultaneously.

The Credential Cascade That Reached LiteLLM

The LiteLLM compromise did not originate with LiteLLM's own infrastructure. It was the fourth wave of a coordinated supply chain campaign by TeamPCP, a threat actor with a documented history of targeting cloud-native infrastructure [3]. A comprehensive treatment of the full four-wave campaign—spanning Trivy, CanisterWorm, Checkmarx KICS, and LiteLLM—is available in the CSA research note "TeamPCP: CI/CD Security Tool Supply Chain Compromise" (March 25, 2026).

For the purposes of this note, the causal chain begins on March 19, 2026, when TeamPCP compromised the Aqua Security Trivy vulnerability scanner by force-pushing malicious code to the `aquasecurity/trivy-action` and `aquasecurity/setup-trivy` GitHub repositories [3]. The malicious Trivy payload performed memory scraping of CI/CD runner processes, harvesting every secret accessible to the runner—cloud credentials, GitHub PATs, SSH keys, and crucially, secrets stored in `.env` files in the repository. LiteLLM's CI/CD pipeline ran Trivy during the exposure window. The `PYPI_PUBLISH` service account token, stored as a `.env` variable in LiteLLM's repository, was harvested in that scrape [2]. With this token, TeamPCP bypassed LiteLLM's official release workflow entirely and published malicious packages directly to PyPI five days later.

This causal chain illustrates a structural risk that is not specific to LiteLLM: any project whose CI/CD pipeline ran a compromised Trivy or Checkmarx KICS action during the March 19–23 exposure window, and whose repository contained publishing tokens or other sensitive credentials in environment variables, should be considered potentially compromised through the same mechanism [3].

Security Analysis

The Backdoor Mechanics

Malicious LiteLLM versions 1.82.7 and 1.82.8 were published to PyPI at 10:39 and 10:52 UTC on March 24, 2026—a 13-minute interval that reflects active payload iteration between releases. PyPI quarantined both packages at approximately 11:25 UTC, but the effective exposure window extended to approximately 16:00 UTC per LiteLLM's own advisory, as cached versions remained accessible in some environments [1][2][10]. The last safe version is 1.82.6.

The v1.82.7 payload embedded Base64-encoded malicious code at line 128 of `litellm/proxy/proxy_server.py`. This required an active LiteLLM proxy invocation to trigger—a meaningful constraint, since not all LiteLLM deployments use the proxy component. Version 1.82.8 removed that constraint entirely. By adding a `litellm_init.pth` file to the Python `site-packages/` directory, the payload ensured execution on every Python interpreter startup, regardless of whether LiteLLM was ever called [1][10]. The `.pth` mechanism is difficult to detect because it repurposes a legitimate Python feature—used by packaging tools to extend `sys.path`—that may not be covered by endpoint detection rules focused on conventional malware injection patterns, though detection coverage varies by vendor and rule set. Any Python process on the host, including scheduled cron jobs, data pipelines, and administrative scripts, would trigger the exfiltration routine (virtual environments with isolated `site-packages` may have been unaffected depending on where the malicious package was installed).

The payload executed a three-stage sequence. In the first stage, the malware performed recursive credential harvesting up to six directory levels deep across eight paths, collecting SSH private keys, cloud provider credential files (AWS `credentials`, GCP `application_default_credentials.json`, Azure CLI tokens), Kubernetes service account tokens and kubeconfig files, database credentials, TLS private keys, CI/CD secret files, and the contents

of all `.env` files within scope. It supplemented this with `/etc/passwd` and `/etc/shadow` on Linux hosts. Harvested material was compressed into an encrypted archive named `tpcp.tar.gz` and exfiltrated to attacker-controlled infrastructure at `models.litellm[.]cloud` [1][2].

In the second stage, if Kubernetes credentials were present, the payload deployed privileged pods named `node-setup-{node_name}` to the `kube-system` namespace, configured with `hostPID: true`, `hostNetwork: true`, and a full host filesystem bind mount. This enabled lateral movement across the entire Kubernetes cluster, extending the credential harvest to secrets stored in other pods and namespaces [2]. In the third stage, the payload established a persistent systemd service disguised as "System Telemetry Service" that polled attacker infrastructure for additional commands on a 50-minute interval [1]. The malware also checked the host's timezone and locale settings, and executed a recursive filesystem wiper against hosts identified as operating in Iran—a destructive component consistent with TeamPCP's geopolitically targeted behavior observed in the Trivy wave [3].

The AI-Specific Credential Risk

The LiteLLM backdoor exposed a credential category not fully addressed by most incident response frameworks: LLM provider API keys. Unlike a generic credential stealer targeting cloud infrastructure tools, this compromise specifically targeted the API keys and configuration files that AI-using organizations configure alongside LiteLLM. LLM provider API keys—for OpenAI, Anthropic, Google Gemini, AWS Bedrock, and others—are commonly stored in `.env` files and environment variables, a configuration pattern that LiteLLM explicitly supports and that is prevalent in both development and production environments. These keys are the primary authentication mechanism for production AI workloads. Compromise of an LLM API key provides an attacker with: the ability to consume API capacity at the victim organization's expense, access to any context or data submitted to the LLM API through that key, and in some configurations, the ability to manipulate model responses seen by downstream users or systems.

For organizations using LiteLLM in agentic AI pipelines—where the library orchestrates calls to LLMs that then invoke tools, retrieve documents, or take actions on external systems—the risk compounds. A compromised API key in an agentic context is not merely a billing or data exposure issue; it is a potential foothold for manipulating the behavior of autonomous AI systems. An attacker with a valid API key can craft targeted inputs to those systems, test for prompt injection vulnerabilities, or enumerate what data those systems have access to.

The scale of potential exposure warrants emphasis. With 95 million monthly PyPI downloads and broad presence as a transitive dependency, even assuming a small fraction of installations ran v1.82.7 or v1.82.8 during the approximately five-hour exposure window, the absolute number of potentially affected hosts is substantial. BleepingComputer reported approximately 500,000 exfiltration events, though the same source noted many were duplicates from the same compromised host triggering the payload through multiple Python processes [1]. Boost Security identified approximately 70 repositories as confirmed affected in their initial analysis, as reported by tl;dr sec [11]. To our knowledge, the March 2026 incident is the first documented mass-scale supply chain compromise to explicitly expose AI API keys as a targeted credential category, representing a notable expansion of attacker objectives.

The Persistence Artifact Problem

The `.pth` file technique deployed in v1.82.8 presents a specific remediation challenge that distinguishes this incident from supply chain attacks that require active invocation. Even if an organization removes the malicious LiteLLM package and upgrades to a clean version, the `litellm_init.pth` file remains in `site-packages/` unless explicitly removed. If this artifact is not found and deleted, the credential-stealing routine will continue to execute on every Python startup indefinitely, even after the malicious package has been uninstalled. This means standard "update the package" remediation, while necessary, is not sufficient. A targeted search for the persistence artifact is required.

Similarly, the systemd "System Telemetry Service" backdoor survives package removal and continues polling attacker infrastructure for commands. Organizations must audit systemd unit files and disable or remove any service matching the indicators of compromise before considering remediation complete. Kubernetes clusters that were laterally compromised during the second payload stage may have persistent pod or DaemonSet artifacts in the `kube-system` namespace that also require explicit discovery and removal.

Recommendations

Immediate Actions

Any organization that installed LiteLLM v1.82.7 or v1.82.8 should assume the host is fully compromised and begin credential rotation immediately. While v1.82.7 required proxy invocation to trigger, the execution conditions for any specific deployment may not be fully known; erring toward full remediation

is the prudent posture. The first priority is rotating all LLM API keys (OpenAI, Anthropic, Google, AWS Bedrock, Azure OpenAI, Cohere, and any other LLM providers configured in the environment), because these credentials are specific to AI workloads and may not be covered by existing cloud credential rotation runbooks. All cloud provider credentials (AWS IAM, GCP service accounts, Azure service principals), SSH keys, Kubernetes service account tokens, database credentials, and any other secrets present in `.env` files or environment variables on the affected host must also be rotated immediately.

Following credential rotation, scan all Python `site-packages/` directories on affected systems for `litellm_init.pth` and remove it. Check for `~/.config/sysmon/sysmon.py` and inspect systemd unit files for services named "System Telemetry Service," "pgmon.service," or "sysmon.service." Audit Kubernetes clusters for pods named `node-setup-*` in the `kube-system` namespace and for DaemonSets with universal tolerations and `hostPID: true` or `hostNetwork: true` flags. Search for `/tmp/pglog` and `/tmp/.pg_state` on affected Linux hosts. Block outbound connections to `models[.]litellm[.]cloud` and `checkmarx[.]zone` (83.142.209.11) at the network perimeter.

Organizations that cannot confirm which LiteLLM version was installed—due to absent dependency lock files or cached package availability—should treat the host as potentially compromised and proceed with credential rotation regardless.

Short-Term Mitigations

Dependency pinning with hash verification is the most direct technical control for preventing this class of attack. Using a `requirements.txt` with pinned version numbers is insufficient; version tags on PyPI are mutable only in the sense that a new malicious release can publish under a new version number, but a known-good release cannot be retroactively altered. The defense is to pin to specific package hashes using `pip install --require-hashes` or an equivalent mechanism in the project's dependency management toolchain. This ensures that even if a new malicious version is published at a higher version number, automated dependency update tooling cannot silently pull it unless a maintainer explicitly approves the new hash [5].

LLM API keys should be stored in dedicated secrets management systems—AWS Secrets Manager, HashiCorp Vault, GCP Secret Manager, Azure Key Vault—rather than in `.env` files in source repositories or on CI/CD runner filesystems. Short-lived credentials, scoped to specific workloads and rotated automatically, substantially reduce the blast radius of any credential harvesting event. For AI workloads specifically, LLM provider API key rotation is often not included in standard incident response runbooks because these keys are frequently treated as application configuration rather than security-sensitive credentials. This gap should be closed.

Introduce software composition analysis tooling that continuously monitors installed packages for known-malicious versions, sourcing advisory data from PyPA's advisory database (which assigned PYSEC-2026-2 to this incident) and from commercial SCA vendors. Several SCA vendors had detections available within hours of initial reporting [1][2]; organizations with those tools integrated into deployment pipelines may have received alerts before installation completed, depending on tool configuration and vendor detection timelines.

Strategic Considerations

Prior to March 2026, supply chain attacks against widely adopted, legitimate AI framework packages were uncommon; incidents in the AI Python ecosystem were primarily typosquatting attacks against lower-profile or newly created packages rather than compromises of established, high-download libraries. The LiteLLM compromise demonstrates that a single poisoned release of a high-download AI toolchain package can reach tens of millions of installations in a matter of hours, and that AI-specific credentials—LLM API keys—constitute a meaningful target category that existing incident response frameworks do not fully address. Together, these factors represent a qualitative shift in supply chain risk for AI-native organizations.

Organizations building AI-native applications should extend their software supply chain security practices to explicitly cover AI toolchain dependencies: LLM libraries, agent orchestration frameworks, vector database clients, and embedding model wrappers. These packages occupy a privileged position in AI infrastructure—they handle API keys, connect to sensitive data sources, and increasingly orchestrate autonomous agents with broad tool access. The security bar applied to them should be commensurate with that privileged position, not treated as a category apart from the application security program.

The TeamPCP campaign's demonstrated use of credential cascading—where tokens harvested from one victim enable attacks against organizations that interacted with the first victim's tools—suggests that organizations should evaluate their exposure to second-order supply chain risk. If a security tool used in your CI/CD pipeline is compromised, the attacker inherits access to every secret that pipeline touches. This argues for OIDC-based short-lived credential issuance wherever possible, strict least-privilege scoping of CI/CD credentials, and treatment of CI/CD runner memory as a high-value attack surface.

The timing of the TeamPCP campaign overlapped with major security conferences—a period when defender attention may be divided. Whether this was deliberate is unconfirmed, but organizations should ensure that incident response capabilities are not degraded during major industry events, and that automated alerting from SCA and credential monitoring tools does not depend on manual triage to surface critical signals.

CSA Resource Alignment

The LiteLLM supply chain compromise connects directly to CSA's MAESTRO threat modeling framework, which addresses the security of agentic AI systems across seven layers of the AI stack. Layer 2 (Data Operations) and Layer 3 (Agent Execution Environment) are directly implicated: LiteLLM operates as the execution environment intermediary between agentic systems and LLM providers, and a compromised LiteLLM installation corrupts this trust boundary. MAESTRO's threat modeling guidance for execution environment integrity applies directly to the validation and monitoring of AI toolchain dependencies, and organizations using MAESTRO to assess their agentic AI deployments should now explicitly include LLM library supply chain risks in their threat models [6].

CSA's AI Controls Matrix (AICM) addresses dependency integrity for AI systems, an area directly activated by this incident. The AICM's controls on AI model and component supply chain security apply to the libraries and toolkits that AI systems depend on, not merely to model weights or training data. The LiteLLM incident illustrates that a compromised AI support library can expose every credential the AI system uses, including the LLM API keys that are the keys to the AI kingdom for the applications built on them [7].

The CSA Cloud Controls Matrix (CCM) domain AIS (Application & Interface Security), specifically controls AIS-04 and AIS-07, requires security assessment of third-party software components as part of a mature application security program. These controls should be explicitly extended to cover AI toolchain dependencies. The CCM's SEF (Security Incident Management) domain is activated for any organization that installed malicious LiteLLM versions, as mandatory incident notification and documentation requirements apply where credential exposure has occurred [8].

CSA's Zero Trust guidance is directly applicable to the credential architecture failure this incident exposed. The `.env` file storage of LLM API keys and PyPI publish tokens represents a standing credential posture: long-lived secrets persisting in accessible filesystem locations, available to any process with read access to the environment. Zero Trust's principle of continuous verification, operationalized through secrets management systems with short-lived, scoped credentials and OIDC-based authentication, eliminates the class of credential at rest that this attack harvested. Adopting Zero Trust credential practices for AI toolchain secrets specifically should be a direct actionable output of this incident for AI-native organizations [9].

CSA's guidance on software supply chain transparency, including SBOM adoption per CycloneDX and SPDX standards, provides the inventory foundation required to respond rapidly to future incidents of this kind. Organizations with current SBOMs covering their AI toolchain dependencies would have been able to identify affected systems more rapidly; without them, manual package auditing across environments substantially extends the exposure window and delays accurate exposure assessment.

References

[1] BleepingComputer, "Popular LiteLLM PyPI Package Backdoored to Steal Credentials, Auth Tokens," March 2026. https://www.bleepingcomputer.com/news/security/popular-litellm-pypi-package-compromised-in-teampcp-supply-chain-attack/

[2] Wiz, "Threes a Crowd – TeamPCP Trojanizes LiteLLM in Continuation of Campaign," March 2026. https://www.wiz.io/blog/threes-a-crowd-teampcp-trojanizes-litellm-in-continuation-of-campaign

[3] Cloud Security Alliance AI Safety Initiative, "TeamPCP: CI/CD Security Tool Supply Chain Compromise," March 25, 2026.

[4] Endor Labs, "TeamPCP Isn't Done: Threat Actor Behind Trivy and KICS Compromises Now Hits LiteLLM's 95 Million Monthly Downloads on PyPI," March 24, 2026. https://www.endorlabs.com/learn/teampcp-isnt-done

[5] PyPA, "PYSEC-2026-2: Malicious code in litellm," March 24, 2026. https://github.com/pypa/advisory-database/blob/main/vulns/litellm/PYSEC-2026-2.yaml

[6] Cloud Security Alliance, "Agentic AI Threat Modeling Framework: MAESTRO," February 2025. https://cloudsecurityalliance.org/blog/2025/02/06/agentic-ai-threat-modeling-framework-maestro

[7] Cloud Security Alliance, "AI Controls Matrix (AICM)," 2025. https://cloudsecurityalliance.org/artifacts/ai-controls-matrix/

[8] Cloud Security Alliance, "Cloud Controls Matrix v4," 2021. https://cloudsecurityalliance.org/research/cloud-controls-matrix/

[9] Cloud Security Alliance, "Software-Defined Perimeter and Zero Trust," 2022. https://cloudsecurityalliance.org/research/topics/zero-trust/

[10] LiteLLM, "Security Update March 2026," March 24, 2026. https://docs.litellm.ai/blog/security-update-march-2026

[11] tl;dr sec, "Issue #321: LiteLLM & Security Scanner Supply Chains Compromised," published approximately March 27, 2026. https://tldrsec.com/p/tldr-sec-321

[14] JFrog Security Research, "TeamPCP Strikes Again – Telnyx Popular PyPI Library Compromised," March 27, 2026. https://research.jfrog.com/post/team-pcp-strikes-again-telnyx-popular-library-hit/

Further Reading

[12] Risky Business, "Episode #830 — LiteLLM and Security Scanner Supply Chains Compromised," March 2026. https://risky.biz/RB830/

[13] Andrej Karpathy, "Software horror: litellm PyPI supply chain [attack]," X (formerly Twitter), March 24, 2026.