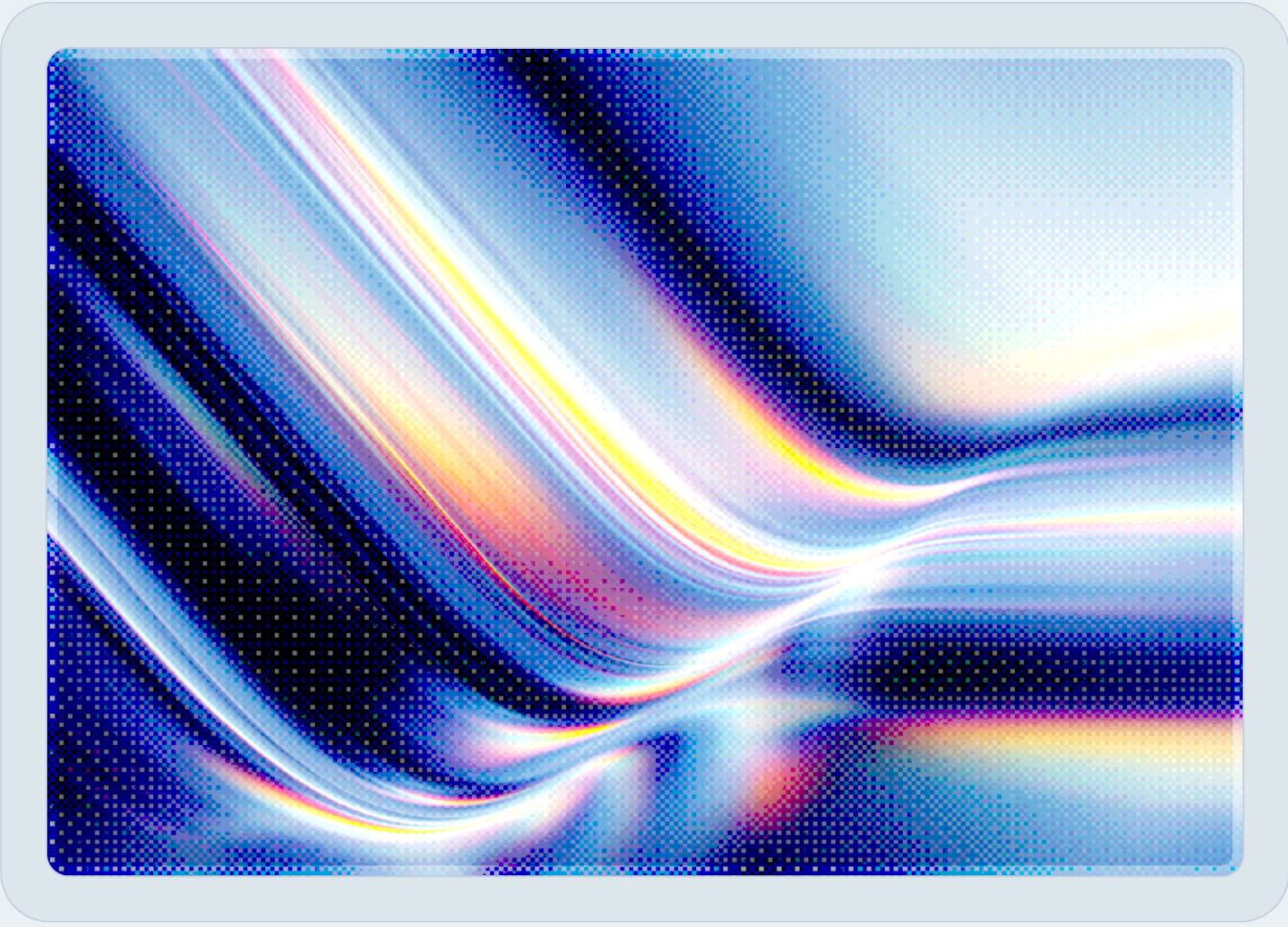
CSAI Foundation | Cloud Security Alliance

AI Developer Supply Chain Under Siege

Quasar Linux RAT and PCPJack Target Model Publishing Pipelines

2026-05-11

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Table of Contents

Executive Summary

The software supply chain has always been a high-value target for sophisticated threat actors, but the rapid growth of AI development has introduced a new category of victim: the machine learning engineer, data scientist, and MLOps practitioner whose workstation holds the keys to model registries, AI gateway libraries, and inference infrastructure serving millions of downstream users. Two campaigns disclosed in spring 2026 illustrate this threat with unusual specificity.

Quasar Linux RAT—designated QLNX by Trend Micro researchers—is a previously undocumented Linux implant engineered specifically for silent, persistent access to developer systems [1]. It combines a fileless execution strategy, a dual-layer rootkit using both LD_PRELOAD hooks and eBPF kernel instrumentation, a Pluggable Authentication Module backdoor, and a credential harvester that sweeps for every token type relevant to software publishing and cloud deployment. PCPJack, identified by SentinelLABS in late April 2026, operates differently: it is a self-propagating cloud worm that exploits five distinct CVEs to spread across exposed infrastructure, evicts the competing TeamPCP malware cluster, and harvests credentials from cloud-native services including RayML—a distributed compute framework used extensively in AI workloads [5]. Both campaigns demonstrate that AI development pipelines have become primary targets rather than incidental casualties of broader cybercriminal activity.

The convergence of these campaigns with the March 2026 TeamPCP-attributed compromise of LiteLLM—a Python package that proxies calls to large language model APIs and accumulates 3.4 million daily downloads—illustrates precisely what credential theft from a developer workstation or CI/CD pipeline can yield [9]. A single stolen PyPI upload token enabled the injection of a malicious .pth file that would execute automatically within every Python process on every downstream installation. The AI infrastructure that enterprises are racing to deploy is being compromised at the point of publication, before it ever reaches the organizations that trust it.

This paper provides a technical analysis of QLNX and PCPJack, examines the LiteLLM incident as a case study in pipeline takeover, and offers practitioners a structured framework for hardening the AI developer supply chain. Recommendations are grounded in CSA's MAESTRO agentic AI threat modeling framework, the AI Controls Matrix, the Cloud Controls Matrix, and Zero Trust guidance.

1. Introduction and Background

1.1 The AI Development Pipeline as Attack Surface

Modern AI development does not resemble the stereotypical image of data scientists training models in isolation. Production AI systems are built and maintained through industrialized pipelines that closely mirror conventional software DevOps: source code is managed in Git repositories, dependencies are pulled from PyPI and npm, container images are built and pushed to registries, model weights are published to platforms such as Hugging Face, and inference infrastructure is deployed on Kubernetes clusters spanning multiple cloud providers. At each stage of this pipeline, automated credentials authorize the work. A token stored in a developer's `.pypirc` file can publish a new version of a library to PyPI. A key in `.aws/credentials` can provision GPU compute. An API key in a CI/CD environment variable can invoke a large language model inference endpoint with billing implications measured in thousands of dollars per hour.

This architecture creates a structural problem that threat actors have recognized and are now actively exploiting. The same credentials that enable legitimate development workflow also authorize malicious publishing if stolen. Unlike exploiting a vulnerability in deployed software—which may require discovering a specific code path—stealing developer credentials offers a pre-authorized path to supply chain compromise. An attacker who obtains a package maintainer's PyPI token does not need to bypass any technical control; they simply authenticate as the maintainer.

The risk is compounded by the profile of the typical AI developer's workstation. Machine learning practitioners increasingly favor Linux environments, in part because the toolchains for GPU compute, distributed training frameworks, and containerized experimentation have historically matured first on that platform. They maintain credentials for an unusually large number of services—model registries, cloud providers, experiment tracking platforms, vector databases, and multiple LLM API providers—because their work requires integrating all of them. Developer credential hygiene in AI-adjacent tooling has historically been complicated by default configurations— `.pypirc`, `.npmrc`, and `.aws/credentials` all store secrets in plaintext by default—and there is limited industry evidence of widespread token rotation practices across these environments, creating structural exposure that persists independent of individual practitioners' security awareness.

1.2 The Threat Actor Ecosystem in Early 2026

The two campaigns analyzed in this paper did not emerge in isolation. They are products of a threat actor ecosystem that has been professionalizing its targeting of developer infrastructure over the preceding two years. TeamPCP, the cluster attributed to the LiteLLM compromise and whose infrastructure PCPJack

subsequently colonized, conducted "several high-profile supply chain intrusions throughout early 2026" according to SentinelLABS [5]. The group demonstrated operational sophistication by targeting CI/CD pipeline dependencies—specifically a Trivy security scanning GitHub Action—gaining privileged access to publishing credentials without requiring workstation-level compromise. PCPJack's subsequent emergence, with tooling that first evicts TeamPCP artifacts before installing its own, suggests either an internal fracture within that cluster or competition between criminal operators for access to the same victim pool [5].

QLNX represents a separate development track: a mature, custom-built Linux implant whose depth of engineering—58 distinct C2 commands, multi-mechanism persistence, dynamic on-host compilation, P2P mesh networking—indicates investment beyond opportunistic criminal tooling [1]. The malware does not include ransomware, cryptominer, or destructive payloads. Its entire post-compromise activity is oriented toward credential accumulation and maintaining persistent, undetected access—precisely the profile of a capability being operated for supply chain leverage rather than immediate monetization.

2. Quasar Linux RAT (QLNX): Technical Analysis

2.1 Discovery and Initial Assessment

Trend Micro researchers disclosed QLNX in May 2026 as a previously undocumented Linux remote access trojan targeting software developers and DevOps engineers [1][3][4]. The designation "Quasar Linux" is distinct from the older Windows-based Quasar RAT family; the two do not share a codebase and the naming appears coincidental. QLNX represents a ground-up engineering effort aimed squarely at the Linux developer workstation, with capabilities selected specifically to enable supply chain compromise rather than general cybercrime.

The malware's technical architecture is notable for the depth of its evasion strategy, the breadth of its credential harvesting scope, and several engineering choices that reflect awareness of enterprise detection capabilities. Understanding each dimension of its design is essential for defenders who must identify and disrupt it.

2.2 Fileless Execution and Memory Residence

Upon initial execution, QLNX copies itself into an anonymous in-memory file created through the Linux kernel's `memfd_create` interface, re-executes from that memory copy, and deletes the original on-disk binary [1][16]. The result is a running process with no corresponding file on the filesystem that standard endpoint tools would detect through file scanning or file integrity monitoring. The process then masquerades its name to resemble a legitimate kernel thread—using identifiers such as `kworker` or `ksoftirqd` that appear routinely in system process listings—making casual inspection of running processes unrevealing [2].

This fileless approach is accompanied by one of QLNX's more distinctive engineering choices: the malware carries its own C source code as embedded string literals within the binary [1]. When it needs to compile additional components or re-establish itself after partial removal, it extracts and compiles this source code directly on the target host using the system's own `gcc` toolchain. This means QLNX does not need to download compiled binaries—which might trigger network-based detection—and produces fresh executables that will differ in hash from any previously observed sample. For defenders relying on static signature matching, this design creates a perpetual evasion challenge.

2.3 Dual-Layer Rootkit Architecture

QLNX deploys two distinct rootkit mechanisms that operate at different privilege levels, providing defense-in-depth against detection [1][21]. The first is a userland rootkit implemented through LD_PRELOAD shared library injection. By inserting a malicious shared library into the process environment, QLNX intercepts calls to core libc functions—specifically those used to enumerate files and processes—and filters results to hide its own presence. Directory listings, process enumeration tools, and file integrity checks that work through standard library calls will return sanitized results that omit QLNX's artifacts.

The second layer operates at the kernel level through Extended Berkeley Packet Filter (eBPF) programs loaded into the Linux kernel. The eBPF rootkit controller manages kernel-level BPF maps that conceal specific process IDs, file paths, and network ports from kernel-space visibility [1]. Because eBPF programs run with kernel privileges and operate below the layer where most security tools observe behavior, this mechanism can evade endpoint detection that the LD_PRELOAD layer might not defeat. Together, the two layers create overlapping coverage: an investigator who detects and removes the userland rootkit still faces the eBPF component, and vice versa.

The combination reflects an understanding that modern Linux endpoint security operates at multiple visibility levels. A single evasion layer is increasingly insufficient against products that combine userland monitoring with kernel telemetry. QLNX's dual-layer design appears engineered specifically to defeat this class of defense.

2.4 PAM Backdoor and Authentication Interception

One of QLNX's most operationally significant capabilities is its Pluggable Authentication Module backdoor [1]. The Linux PAM framework is the standard mechanism through which authentication events are handled on Linux systems—including SSH logins, `sudo` elevations, and local console authentication. By inserting an inline hook into the PAM stack, QLNX intercepts authentication events at the point where credentials are presented in plaintext, before they are hashed or otherwise protected.

The practical consequence is that any user who authenticates to a QLNX-compromised system transmits their plaintext credentials to the attacker's command-and-control infrastructure, regardless of whether the authentication itself succeeds or fails. SSH session data—including keystrokes entered during remote sessions—is also logged and exfiltrated. A second PAM-based credential logger is automatically loaded into every dynamically linked process on the system [1], creating a comprehensive authentication interception capability. For organizations where developers access shared build servers or CI/CD infrastructure via SSH, a single QLNX infection can harvest credentials for every user who subsequently connects to that system.

2.5 Developer Credential Harvesting

The credential harvesting module represents the clearest evidence of QLNX's developer-specific targeting intent. Rather than pursuing generic browser credentials or cryptocurrency wallet files—the typical focus of commodity infostealers—QLNX systematically seeks out the configuration files and token stores that underpin software development and cloud operations [1][2]. The harvester targets the following credential types, each of which maps directly to a publishing or deployment capability:

Credential File	Target Service	Publishing or Deployment Capability
`.npmrc`	npm registry	Publish JavaScript packages
`.pypirc`	Python Package Index	Publish Python packages to the world's most widely used language registry
`.git-credentials`	Git repositories	Authenticated push access to source code repositories
`.aws/credentials`	Amazon Web Services	Provision cloud infrastructure; access S3 model artifact storage
`.kube/config`	Kubernetes clusters	Deploy containerized applications and inference services
`.docker/config.json`	Container registries	Push container images
`.vault-token`	HashiCorp Vault	Access secrets management infrastructure
Terraform state and credential files	Infrastructure as Code	Provision cloud resources across providers
GitHub CLI authentication tokens	GitHub	Interact with repositories and CI/CD pipeline secrets
`.env` files and environment variable stores	Various	Expose application secrets and API keys in bulk

In combination, these stolen credentials would provide an attacker with authorization to inject malicious content at multiple points in an AI development and deployment pipeline—from source code through published packages, container images, and cloud infrastructure configuration.

2.6 Command-and-Control and Persistence

QLNX's C2 framework implements 58 distinct commands that provide operators with comprehensive remote control capabilities, including file manipulation, process management, network tunneling, system profiling, and exfiltration tasking [1][2]. The infrastructure supporting these commands uses peer-to-peer mesh networking, allowing compromised systems to relay communications to one another rather than depending entirely on centralized servers. This design provides operational resilience: taking down C2 server infrastructure does not terminate operator access if infected hosts can continue routing traffic through other infected nodes.

For persistence, QLNX supports at least seven distinct mechanisms, including crontab entries, desktop autostart entries, init scripts, systemd service files, and `.bashrc` shell injection [2]. Using multiple persistence mechanisms simultaneously means that remediation efforts that address only one mechanism—for example, removing a rogue systemd service—leave the infection functional through the others. QLNX also profiles the host environment to detect containerized execution contexts, adjusting its behavior to remain viable across both bare-metal and virtualized deployment scenarios.

3. PCPJack: Cloud Infrastructure Worm

3.1 Discovery and Context

SentinelLABS identified PCPJack on April 28, 2026, when a Kubernetes-focused hunting rule on VirusTotal flagged a shell script with unusual initial behavior [5][7]. The script's first actions were not to install malware but to evict and delete the tooling of a competitor: artifacts associated with the TeamPCP threat cluster were systematically removed before PCPJack installed its own capabilities. This behavior—one malware family displacing another upon infection—indicates competition between criminal operators for access to the same population of vulnerable hosts.

The discovery came approximately five weeks after the TeamPCP-attributed LiteLLM compromise on March 24, 2026. While the precise relationship between PCPJack and TeamPCP remains under investigation, SentinelLABS assesses that "PCPJack could be the work of a former member of TeamPCP who is familiar with the group's tradecraft" [5]. The operational knowledge embedded in PCPJack's design—particularly its ability to identify and cleanly remove TeamPCP artifacts—supports the inference of a close relationship between the clusters, whether as defection, splinter group, or hostile takeover of a shared victim pool.

3.2 Propagation Mechanism and Target Discovery

PCPJack's propagation strategy is notable for avoiding hardcoded target lists, which would both limit its reach and create obvious forensic artifacts. Instead, the worm downloads hostname data from Common Crawl parquet files [5][6]. Common Crawl is a nonprofit organization that continuously crawls the public web and makes its data freely available, originally for use in natural language processing research and large language model pre-training. PCPJack repurposes this open-source internet mapping data as a dynamic scanning target list, accessing up to 104 million potential host entries per cycle without any centralized coordination infrastructure [6].

This design means the worm's target discovery capability scales to hundreds of millions of potential hosts per cycle without requiring any centralized coordination infrastructure that defenders could disrupt through traditional takedown mechanisms. Organizations can, however, limit exposure by blocking egress connections to Common Crawl infrastructure at the network perimeter.

The bootstrap process begins with a shell script that configures the payload staging environment, downloads next-stage Python tooling, establishes persistence, and deletes itself—removing forensic evidence of the initial infection vector [5]. A main orchestrator script then launches purpose-built modules for credential theft, vulnerability exploitation, and lateral movement.

3.3 Vulnerability Exploitation

PCPJack exploits five distinct CVEs to gain initial access and spread across networks [6]. CVE-2025-55182, designated React2Shell, is a CVSS 10.0 pre-authentication remote code execution vulnerability in React Server Components arising from unsafe deserialization in the React Server Components Flight protocol [15]. The critical severity rating and pre-authentication nature of this flaw make it particularly valuable for initial access. CVE-2025-29927, CVE-2026-1357 (an unauthenticated file upload vulnerability in WPVivid Backup), CVE-2025-9501, and CVE-2025-48703 round out the exploitation portfolio [6][8]. The breadth of this CVE set reflects a strategy of opportunistic exploitation: any exposed host running one of these vulnerable components is a potential entry point.

Once initial access is established, a propagation module designated "csc" within the PCPJack toolset performs the exploitation work, scanning discovered hosts for vulnerable services and deploying payloads on successful exploitation [5]. The modular Python architecture means individual components can be updated independently, allowing operators to add new CVE exploits as public disclosures occur.

3.4 Cloud-Native and AI Infrastructure Targeting

PCPJack's credential theft modules target a set of cloud-native services that maps closely to the infrastructure stack used in AI development and deployment [5][6][18]. Docker API endpoints, Kubernetes API servers, Redis instances, and MongoDB deployments are all explicitly targeted. Critically, RayML—the distributed computing framework widely used for large-scale machine learning training, hyperparameter optimization, and inference serving—is targeted through a weaponized job submission mechanism [5][17].

A second toolset, identified during the SentinelLABS investigation, extends credential theft to a set of services specifically relevant to AI operations: Anthropic API keys, OpenAI API keys, Google API credentials, Grafana Cloud monitoring tokens, HashiCorp Vault secrets, and OnePassword vault access [5]. The presence of both Anthropic and OpenAI API credentials in this expanded harvest scope indicates a deliberate targeting of organizations operating production AI workloads—these credentials represent both direct financial value (billing exposure on inference API usage) and intelligence value (access to prompts, outputs, and fine-tuning data associated with commercial AI applications).

Unlike many cloud-focused malware families, PCPJack does not deploy cryptominers on compromised infrastructure [5]. The absence of cryptomining—a common monetization path for actors with cloud compute access—and the heavy emphasis on credential collection suggests a business model focused on credential resale, subsequent supply chain attacks using stolen publishing tokens, fraud, or targeted extortion of organizations whose secrets have been exfiltrated.

3.5 Command-and-Control Infrastructure

PCPJack uses Telegram as its primary command-and-control channel, posting host telemetry and credential dumps to one channel while polling a separate channel for operator commands embedded in pinned messages [5][8]. This design exploits Telegram's encrypted messaging infrastructure and the relative difficulty of obtaining takedowns for channels on major commercial messaging platforms—properties that make Telegram a preferred C2 channel across multiple malware families. The approach also reduces the operational burden of maintaining dedicated C2 servers, which are vulnerable to domain takedowns and IP blocklisting.

Data exfiltration employs ephemeral X25519 key pairs for key exchange and ChaCha20-Poly1305 for symmetric encryption when the cryptographic dependencies are available on the target [5][8]. However, if those dependencies cannot be loaded—a condition that may arise on minimal or hardened installations—PCPJack falls back to transmitting credential data in plaintext. This fallback behavior means that some proportion of PCPJack-compromised hosts are transmitting highly sensitive credentials over unencrypted channels, creating an additional exposure vector beyond the intended one.

4. The LiteLLM Compromise: A Pipeline Takeover Case Study

4.1 Background and Significance

LiteLLM is a Python library that provides a unified interface for calling large language model APIs across dozens of providers—OpenAI, Anthropic, Google Gemini, Cohere, and others—through a standardized OpenAI-compatible interface [9][10]. Its architecture makes it a critical piece of AI infrastructure for organizations that wish to abstract their applications from specific model providers, implement load balancing across providers, or maintain fallback chains when primary providers experience outages. As of the March 2026 incident, LiteLLM was receiving approximately 3.4 million daily downloads from PyPI [10]. That scale made it a high-value supply chain target: a poisoned LiteLLM package would be installed automatically by enterprise dependency managers in AI-focused organizations worldwide.

4.2 Attack Vector: The Poisoned Security Scanner

The compromise demonstrates a supply chain attack pattern of increasing sophistication: rather than targeting the primary package maintainer's credentials directly, TeamPCP compromised a dependency of LiteLLM's CI/CD security scanning workflow [9][10]. Specifically, the attack vector was a Trivy GitHub Action—a container image security scanning tool integrated into LiteLLM's automated pipeline to detect vulnerable dependencies before release. By compromising the Trivy dependency used in the scanning workflow, TeamPCP gained execution within LiteLLM's CI/CD environment, where it could observe, exfiltrate, and ultimately abuse the publishing credentials stored as pipeline secrets [9].

This attack vector illustrates a recursive supply chain risk: the tools organizations deploy to secure their supply chains are themselves components of that supply chain. Security scanning tools, code signing utilities, and dependency audit frameworks all require elevated permissions within CI/CD environments, making them high-value targets for adversaries who understand DevOps architecture. Snyk's analysis of the incident described this as a "poisoned security scanner" attack and noted that the trust organizations extend to security tooling is precisely what makes it such an effective attack vector [19].

4.3 Malicious Payload and Execution Mechanism

On March 24, 2026, two LiteLLM releases were published to PyPI with malicious payloads. Version 1.82.7 delivered its payload via malicious code injected into `proxy_server.py`, which required active import of the module to execute. Version 1.82.8 introduced a more severe vector: a malicious `.pth` file named `litellm_init.pth` [11][12][22]. Python `.pth` path configuration files are automatically processed by the Python interpreter during startup for every Python environment in which the package is installed. Unlike most malware injection techniques that require the victim to execute a specific function or import a module, `.pth`-based payloads execute unconditionally whenever Python starts—including in subprocesses, automated scripts, and batch inference jobs that never directly invoke LiteLLM functionality [12].

The payload was designed to harvest AWS tokens, GCP and Azure credentials, SSH keys, and Kubernetes configuration files, then exfiltrate them to attacker-controlled infrastructure [10]. The authorization those credentials provided would enable lateral movement into cloud environments, access to AI infrastructure, and potentially further supply chain compromises using the stolen credentials of organizations that had trusted the LiteLLM package.

According to PyPI's official incident report, the poisoned versions remained available for approximately 2 hours and 32 minutes from first publication to quarantine—a timeline materially longer than some initial reporting suggested [24]. At average pull rates for a package receiving 3.4 million daily downloads, that window represents roughly 350,000 automated downloads, concentrated among CI/CD pipelines running during US business hours. LiteLLM subsequently released version 1.83.0 through an isolated CI/CD v2 pipeline with strengthened security gates, including separate environments for build and release and more granular secret access controls [11].

4.4 Implications for AI Infrastructure Trust

The LiteLLM incident is notable not only for its technical execution but for what it reveals about trust assumptions in the AI development ecosystem. Organizations deploying AI applications implicitly trust that the LLM gateway libraries, prompt management frameworks, vector database clients, and model serving utilities they install from public repositories are unmodified from their authors' intent. That trust is institutionalized through automated dependency management: as documented in industry supply chain security research, few organizations manually audit every version update of every transitive dependency before deployment. The economics of software development at scale depend on that trust being warranted.

TeamPCP's operation demonstrated that the trust relationship can be subverted at the point of publication rather than through vulnerabilities in the published software itself. The attack required no novel exploitation technique; it required only stolen publishing credentials and access to PyPI's global distribution network. For

organizations whose AI applications process sensitive data, invoke paid inference APIs, or make decisions with real-world consequences, this represents a qualitative risk that technical vulnerability scanning does not address.

5. Attack Surface Analysis: AI Development Pipelines

5.1 The Credential-Centric Attack Model

Both QLNX and PCPJack, and the TeamPCP operation to which they are linked, share a common strategic logic: the authoritative path to supply chain compromise in modern software development runs through credentials rather than code. Traditional supply chain attacks—trojanizing open-source libraries directly, compromising version control systems, or tampering with build artifacts in transit—require overcoming progressively strengthened technical controls: code signing, reproducible builds, SLSA framework attestations, and binary transparency mechanisms. Credential theft bypasses many of these controls, because it provides legitimate authorization to perform the malicious publishing action. Attestation-based controls—Sigstore signatures, SLSA provenance records, and PyPI Trusted Publisher tokens—are specifically designed to survive this threat model: a stolen upload token does not grant access to the attestation signing infrastructure used in properly configured pipelines, meaning a stolen-credential publication will fail downstream attestation verification. This makes attestation adoption the highest-priority compensating control against credential-based supply chain attacks.

The AI development ecosystem has amplified the underlying risk along several dimensions. The proliferation of specialized platforms—model registries, experiment tracking services, vector database providers, LLM API providers—has expanded the credential surface that each AI developer must manage. Credentials that would previously have been unnecessary (PyPI tokens, Hugging Face write tokens, Ray cluster API keys) are now routinely stored in developer home directories and CI/CD pipeline secrets. The depth of that credential portfolio, concentrated on the Linux workstations that AI practitioners prefer, represents a substantial attack surface.

5.2 CI/CD Pipeline as the Highest-Leverage Target

Within the AI development pipeline, CI/CD infrastructure occupies a structurally privileged position that makes it the highest-leverage target for credential-seeking threat actors. Build pipelines operate with broader permission scopes than individual developer workstations—they may hold cloud provider credentials for multiple environments, container registry push access, package signing keys, and deployment credentials —and they execute continuously without human oversight. A secret exfiltrated from a CI/CD environment may not be discovered until it is used maliciously, because legitimate automated workflows using the same credential create baseline noise that obscures anomalous usage.

The LiteLLM compromise demonstrates how a single dependency in the security scanning layer—a component specifically deployed to detect supply chain risks—can expose the entire credential set of a build pipeline. This pattern is not limited to Trivy; any GitHub Action, CI/CD plugin, or build dependency that executes within the privileged pipeline environment and that has been compromised represents the same risk. The attack surface analysis must extend beyond first-party code to every component that runs with pipeline-level access.

5.3 The RayML and AI Compute Surface

PCPJack's explicit targeting of RayML infrastructure through weaponized job submissions represents an emerging attack surface that deserves specific attention [5]. Ray is a distributed computing framework used extensively in AI development for parallel hyperparameter searches, distributed model training, and large-scale inference serving. Ray clusters operate with broad access to the compute infrastructure—often including GPU nodes, high-memory VMs, and attached storage—and the Ray job submission API may be exposed within internal networks without requiring strong authentication.

An attacker who gains the ability to submit Ray jobs to a cluster can execute arbitrary code within the cluster's permission context, potentially accessing training data, model weights, experiment results, and any credentials stored in the cluster environment. In organizations where Ray clusters are shared across multiple teams, a job submission attack can expand from the initial foothold to comprehensive access across the shared infrastructure. The PCPJack campaign indicates that this attack surface is not merely theoretical—it is being actively exploited.

5.4 API Credential Theft and LLM Provider Access

The second-stage PCPJack toolset's collection of Anthropic and OpenAI API credentials introduces a dimension of risk that extends beyond supply chain integrity into operational AI security [5]. Organizations that deploy AI applications backed by commercial LLM APIs typically store those API keys in environment variables, Kubernetes secrets, or secrets management services accessible from their application runtime. PCPJack's targeting suggests that actors are accumulating these credentials both for immediate financial exploitation—using stolen API keys to make inference calls billed to the victim—and potentially for more sophisticated purposes.

Stolen LLM API credentials could enable exfiltration of proprietary prompts and fine-tuning data, unauthorized access to model customizations, or injection of adversarial inputs into applications for which the attacker now holds valid credentials. For organizations that have embedded proprietary knowledge, customer data, or competitive intelligence into AI system prompts or fine-tuned models accessible through these API keys, a credential compromise represents a confidentiality breach with implications well beyond compute billing fraud.

6. Threat Actor Dynamics and Attribution

6.1 TeamPCP and the Emerging AI Supply Chain Threat Cluster

TeamPCP established itself through early 2026 as a threat actor persona specifically oriented toward supply chain attacks on developer infrastructure [10]. The group demonstrated operational sophistication by successfully targeting CI/CD pipeline dependencies—specifically a Trivy security scanning GitHub Action—gaining privileged access to publishing credentials without requiring workstation-level compromise. The LiteLLM compromise, along with reported compromises of the Telnyx communications library, demonstrates an intent to target AI infrastructure libraries specifically [10].

Datadog Security Labs' analysis of the LiteLLM incident identified connections between TeamPCP and the broader malware campaign, attributing the poisoned package publications to the group [10]. The use of a `.pth` file as the payload delivery mechanism in version 1.82.8 is notable because it achieves automatic execution without modifying any imported package code, making it resistant to source-level code review. This operational choice—selecting an execution path that bypasses standard import-level inspection—suggests familiarity with Python environment internals beyond commodity cybercriminal tooling.

6.2 PCPJack: Successor or Competitor?

SentinelLABS' assessment that PCPJack may represent a former TeamPCP affiliate operating independently is supported by several behavioral indicators [5]. The toolset demonstrates detailed knowledge of TeamPCP's infrastructure and artifact locations—knowledge that would require either direct involvement with TeamPCP operations or extensive reconnaissance of their victim pool. The decision to evict TeamPCP from compromised hosts before installing PCPJack's own tooling reflects competitive dynamics consistent with organized criminal ecosystems where access to compromised hosts has market value.

The possible fracture within the TeamPCP cluster, if confirmed, has practical implications for defenders. Competitive dynamics between criminal operators—particularly when access to a shared victim pool is at stake—can drive more aggressive propagation strategies, as each actor seeks to maximize coverage before rivals can evict their tooling. PCPJack's deployment of multiple CVE exploits for automated propagation—rather than TeamPCP's more targeted CI/CD dependency approach—may reflect different threat actor priorities or simply a different phase of the operation, but the net effect is a larger pool of compromised hosts from which credentials can be harvested.

6.3 QLNX Attribution

Trend Micro's disclosure of QLNX did not attribute the malware to a specific named threat actor [1]. The technical sophistication of the implant—particularly the dual-layer rootkit, PAM backdoor, and dynamic on-host compilation—suggests development investment inconsistent with opportunistic criminal actors. The malware's laser focus on developer credentials, absence of cryptomining or ransomware capabilities, and P2P mesh networking design are consistent with a threat actor operating for intelligence collection, supply chain leverage, or access-as-a-service—selling persistent access to developer workstations to other threat actors who will use the harvested credentials.

Whether QLNX is connected to the TeamPCP/PCPJack ecosystem or represents a separate operation is not established by current public reporting. The credential types targeted by all three campaigns substantially overlap, suggesting that even if the groups are distinct, they are harvesting for the same downstream market.

7. CSA Framework Alignment

7.1 MAESTRO: Threat Modeling for AI Pipeline Integrity

The Cloud Security Alliance's MAESTRO framework—Multi-Agent Environment, Security, Threat, Risk, and Outcome—provides a seven-layer reference architecture for threat modeling agentic and AI systems [13]. Its explicit treatment of the interactions between layers, and its extension of traditional threat categories to cover AI-specific attack surfaces, makes it directly applicable to the pipeline integrity risks exposed by QLNX and PCPJack.

MAESTRO's Layer 1 (Foundation Models) and Layer 2 (Data Operations) are threatened when supply chain attacks deliver poisoned packages into AI application dependencies or compromise the credentials that authorize model artifact publication. CSA has published guidance applying MAESTRO specifically to CI/CD pipeline threat models [20], which maps directly to the attack patterns described in this paper. Organizations building AI applications should use MAESTRO as the organizing framework for pipeline security reviews, explicitly threat-modeling the credential stores, build dependencies, and deployment authorization paths that QLNX and PCPJack target.

7.2 AI Controls Matrix: Supply Chain and Model Security Domains

The CSA AI Controls Matrix (AICM) v1.0, released in July 2025, establishes 243 control objectives across 18 security domains addressing the full spectrum of AI system security [14]. Two domains are directly implicated by the campaigns analyzed in this paper.

The Supply Chain Management, Transparency, and Accountability domain addresses controls for verifying the integrity of AI components, managing third-party dependencies, and maintaining accountability for published AI artifacts. The controls in this domain should be interpreted in light of the LiteLLM compromise: the poisoned Trivy dependency that enabled the attack was a supply chain component that fell within the scope of CI/CD pipeline supply chain management. Organizations applying AICM controls in this domain should extend their scope explicitly to include build-time dependencies—security scanners, test frameworks, and linting tools—that execute with elevated pipeline permissions.

The Model Security domain addresses controls for protecting model artifacts, authorizing model publishing, and detecting unauthorized modifications to deployed models. The credential theft capabilities of both QLNX and PCPJack target precisely the authorization mechanisms these controls protect. Implementing

strong controls around model registry credentials—including short-lived tokens, publishing action logging, and mandatory code review for version increments—directly addresses the threat actor's intended exploitation path.

7.3 Cloud Controls Matrix: CI/CD and Secret Management

The CSA Cloud Controls Matrix [23] provides relevant controls across several domains applicable to the threat surface exposed by these campaigns. The Supply Chain Management (STA) domain addresses third-party component risk in cloud environments, which should encompass CI/CD pipeline dependencies. The Identity and Access Management (IAM) domain's controls around least privilege, credential rotation, and privileged access management apply directly to the publishing tokens and pipeline secrets that QLNX and PCPJack are designed to steal. The Security Incident Management (SEF) domain's detection and response controls should be updated to include indicators of compromise associated with these campaigns.

7.4 Zero Trust Principles for Developer Infrastructure

Traditional perimeter-based security assumptions are particularly ill-suited to the distributed, credential-heavy architecture of AI development infrastructure. Zero Trust principles—continuous verification, least privilege access, and assumed breach posture—provide a more appropriate foundation for securing developer workstations, CI/CD pipelines, and cloud infrastructure against the credential theft campaigns described in this paper.

Applying Zero Trust to the publishing pipeline means treating every publishing action as requiring explicit re-authorization, even when performed by a recognized identity using valid credentials. Token-level behavioral anomaly detection—flagging publishing events that occur outside established time patterns, from unfamiliar network locations, or for package versions that lack associated code review activity—can detect credential misuse even when the stolen credentials themselves are technically valid. This approach mirrors the zero-trust network access principle that authenticated network sessions still require continuous verification, extended to the credential authorization context of software publishing.

8. Recommendations

8.1 Immediate Actions for Developer Organizations

The most direct protection against QLNX's credential harvesting module is ensuring that developer workstations never hold long-lived publishing tokens in plaintext configuration files. Organizations should migrate PyPI credentials from `.pypirc` files to PyPI's Trusted Publisher mechanism, which uses OIDC-based short-lived tokens generated at pipeline runtime rather than long-lived secrets stored on developer machines. For npm, scoped tokens with publish-only scope should be stored in CI/CD secret management rather than in developer `.npmrc` files. Model registry tokens for Hugging Face and similar platforms should follow the same pattern: no long-lived write tokens on developer workstations, with publishing actions gated through short-lived OIDC credentials.

Container image signing using Sigstore/Cosign and PyPI's attestations feature should be activated for all published packages and images. When downstream consumers verify attestations, a compromised package that fails attestation verification will be rejected before installation—creating a compensating control that can interrupt the kill chain even when a publishing token is stolen. Organizations should also audit their CI/CD pipeline dependencies with the same rigor applied to application dependencies, recognizing that build-time components such as security scanners, linters, and test frameworks execute with the highest privilege level in the pipeline and represent an attack surface that the LiteLLM compromise demonstrated is actively targeted.

8.2 Short-Term Mitigations for AI Infrastructure

For organizations operating Ray clusters, RayML job submission APIs should not be exposed to broad internal network segments without authentication. Ray's native authentication mechanisms—including TLS with client certificates and token-based job API authentication—should be activated, and job submission should be gated through an authorization layer that verifies the submitting identity before accepting workloads. Audit logging of all job submissions, with anomaly detection for unusual submission patterns, provides detection capability against weaponized job submission attacks.

API credentials for LLM providers—Anthropic, OpenAI, Google, and others—should be stored in dedicated secrets management services rather than in environment variable files, application configuration, or CI/CD plaintext variables. Per-application API keys with scoped permissions and usage monitoring allow anomalous billing patterns to be detected quickly. Organizations should establish baseline inference usage profiles for each API key and implement alerting for deviations, treating unexpected usage spikes as a potential indicator of credential compromise.

Developer workstations running Linux should be assessed for QLNX indicators of compromise, including unexpected eBPF programs loaded in kernel space (`bpftool prog list`), PAM configuration modifications, presence of crontab entries for unfamiliar executables, and processes masquerading as kernel threads (`kworker`, `ksoftirqd`) with unexpected parent process relationships. The absence of on-disk artifacts for running processes—detectable through `/proc/[pid]/exe` resolution—is a specific indicator of QLNX's fileless execution.

8.3 Strategic Considerations

At a strategic level, organizations building AI development pipelines should revisit their threat model to explicitly include the developer workstation and CI/CD environment as in-scope assets requiring the same security investment as production systems. The historical tendency to treat developer infrastructure as lower-security than production has created a structural vulnerability that sophisticated threat actors are actively exploiting. The key insight from both QLNX and PCPJack is that compromising a developer machine or CI/CD pipeline may provide more leverage than compromising production infrastructure directly, because the developer holds the credentials that authorize the production deployment.

Software Bill of Materials (SBOM) generation for AI applications should be extended to include build-time dependencies—the components that execute during the CI/CD pipeline rather than those that ship in the final artifact. SBOM data for both build-time and runtime dependencies enables more comprehensive vulnerability tracking and provides a foundation for detecting unexpected component introductions of the type that enabled the LiteLLM compromise. CSA's guidance on Software Transparency and Securing the Digital Supply Chain provides a framework for implementing SBOM practices across the AI development lifecycle.

Endpoint detection for Linux developer workstations should be treated as a security priority equivalent to Windows endpoint protection. The QLNX campaign demonstrates that sophisticated, purpose-built Linux implants targeting developers are now deployed in the wild. Endpoint detection and response products capable of monitoring eBPF program loading, detecting anomalous PAM modifications, and identifying fileless process execution should be deployed on all Linux developer workstations with access to publishing credentials. The incremental cost of extending enterprise EDR coverage to Linux developer workstations is modest relative to the potential regulatory, reputational, and operational impact of a supply chain compromise affecting millions of downstream installations.

9. Conclusions

The QLNX and PCPJack campaigns, viewed alongside the TeamPCP LiteLLM compromise, represent a maturation of supply chain attack methodology that is specifically adapted to the AI development ecosystem. Threat actors have identified that the fastest path to compromising large populations of AI applications is not through vulnerabilities in deployed model weights or inference APIs—it is through the credentials that authorize publishing the libraries those applications depend on. A single stolen PyPI token, a single compromised CI/CD pipeline, can reach hundreds of thousands of downstream installations in the hours before detection and remediation.

The technical sophistication demonstrated by QLNX—fileless execution, dual-layer rootkits combining LD_PRELOAD hooks with eBPF kernel instrumentation, PAM backdoor-based credential interception, dynamic on-host compilation, and P2P mesh networking—indicates investment in a capability designed for long-term, undetected presence on developer workstations. This is not an opportunistic infection seeking quick monetization; it is a strategic capability for accumulating the credentials that enable supply chain leverage. PCPJack's targeting of AI-specific infrastructure—RayML job submission, LLM API keys from both Anthropic and OpenAI—confirms that threat actors have developed detailed understanding of AI development architecture and are exploiting it deliberately.

Organizations building and deploying AI systems must extend their security perimeter to encompass the development pipeline in its entirety. The trust that enterprises and end users place in AI applications flows ultimately from the integrity of the publishing pipeline—the chain of custody from source code to deployed artifact. When that chain is compromised at the credential layer, all downstream security controls operate on a foundation that cannot be trusted. Addressing this risk requires applying Zero Trust principles to publishing authorization, eliminating long-lived credentials from developer workstations, extending supply chain security practices to build-time dependencies, and deploying endpoint security on Linux developer infrastructure with parity to production systems.

CSA's MAESTRO framework, AI Controls Matrix, and Zero Trust guidance provide the structural foundations for this work. The threat is active, the techniques are documented, and the defensive priorities are well-defined.

References

[1] Trend Micro. "Quasar Linux (QLNX) – A Silent Foothold in the Supply Chain: Inside a Full-Featured Linux RAT With Rootkit, PAM Backdoor, Credential Harvesting Capabilities." Trend Micro Research, May 2026.

[2] The Hacker News. "Quasar Linux RAT Steals Developer Credentials for Software Supply Chain Compromise." The Hacker News, May 2026.

[3] BleepingComputer. "New stealthy Quasar Linux malware targets software developers." BleepingComputer, May 2026.

[4] SecurityWeek. "Sophisticated Quasar Linux RAT Targets Software Developers." SecurityWeek, May 2026.

[5] SentinelLABS. "PCPJack | Cloud Worm Evicts TeamPCP and Steals Credentials at Scale." SentinelOne, April/May 2026.

[6] The Hacker News. "PCPJack Credential Stealer Exploits 5 CVEs to Spread Worm-Like Across Cloud Systems." The Hacker News, May 2026.

[7] SecurityWeek. "'PCPJack' Worm Removes TeamPCP Infections, Steals Credentials." SecurityWeek, May 2026.

[8] BleepingComputer. "New PCPJack worm steals credentials, cleans TeamPCP infections." BleepingComputer, May 2026.

[9] Trend Micro. "Your AI Gateway Was a Backdoor: Inside the LiteLLM Supply Chain Compromise." Trend Micro Research, March 2026.

[10] Datadog Security Labs. "LiteLLM and Telnyx compromised on PyPI: Tracing the TeamPCP supply chain campaign." Datadog Security Labs, 2026.

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

[12] Cycode. "LiteLLM Supply Chain Attack: What Happened and How to Respond." Cycode Blog, 2026.

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

[14] Cloud Security Alliance. "AI Controls Matrix." CSA Artifacts, July 2025.

[15] Trend Micro. "CVE-2025-55182: React2Shell Analysis, Proof-of-Concept Chaos, and In-the-Wild Exploitation." Trend Micro Research, 2025.

[16] Security Affairs. "Quasar Linux RAT (QLNX): A Fileless Linux Implant Built for Stealth and Persistence." Security Affairs, May 2026.

[17] GBHackers. "PCPJack Worm Targets Docker, Kubernetes, Redis, and MongoDB Credentials." GBHackers, May 2026.

[18] Dark Reading. "After Replacing TeamPCP Malware, 'PCPJack' Steals Cloud Secrets." Dark Reading, May 2026.

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

[20] Cloud Security Alliance. "Applying MAESTRO to Real-World Agentic AI Threat Models: From Framework to CI/CD Pipeline." CSA Blog, February 2026.

[21] SOC Prime. "Quasar Linux (QLNX): A Supply Chain Foothold with Full RAT Capabilities." SOC Prime, May 2026.

[22] InformationWeek/InfoQ. "PyPI Supply Chain Attack Compromises LiteLLM, Enabling the Exfiltration of Sensitive Information." InfoQ, March 2026.

[23] Cloud Security Alliance. "Cloud Controls Matrix." CSA, current version.

[24] PyPI Blog. "Incident Report: LiteLLM/Telnyx supply-chain attacks." PyPI Blog, April 2026.