TeamPCP: WAV-Embedded Malware Targets AI Developer Toolchains

Audio Steganography and Self-Propagating Worms in the Cascading Supply Chain Attack of March 2026

Unofficial AI-assisted Research

Cloud Security Alliance AI Safety Initiative

2026-03-28

Key Takeaways

- TeamPCP is a confirmed, active threat actor responsible for a cascading supply chain campaign originating February 28, 2026, compromising Trivy, Checkmarx KICS, LiteLLM, and the Telnyx Python SDK across multiple package ecosystems [1][2].
- The campaign's most novel technique — disguising XOR-encrypted, base64-encoded malware payloads inside structurally valid WAV audio containers — allowed credential-stealing executables to bypass network content-type filters and endpoint detection tools focused on conventional executable formats [3][4].
- LiteLLM, an open-source LLM proxy with approximately 95 million monthly PyPI downloads, was compromised with a `.pth` persistence mechanism that executed a credential stealer on every Python process startup, targeting SSH keys, cloud tokens, Kubernetes secrets, and API keys [5][21].
- CanisterWorm, the self-propagating component deployed after the initial Trivy breach, appears to be the first documented supply chain worm to use decentralized Internet Computer Protocol (ICP) blockchain infrastructure for command and control — a technique with significant implications for C2 resilience [7][8].
- All compromised packages have been removed or patched. Organizations that installed affected versions of Trivy, LiteLLM (1.82.7–1.82.8), or Telnyx (4.87.1–4.87.2) should treat all secrets accessible from affected environments as fully compromised and rotate immediately [9][10].

Background

The TeamPCP campaign — tracked under aliases including DeadCatx3, PCPcat, ShellForce, and CipherForce by different threat intelligence vendors — is a hybrid cybercrime and extortion operation [2][14] whose activity has been documented by Arctic Wolf, ReversingLabs, Kaspersky, Sysdig, Sonatype, SANS Institute, and others [1][2][11]. The campaign illustrates how a single initial access event in a security scanning tool can propagate laterally across an ecosystem of dependent projects with cascading credential exposure consequences that outlast the original intrusion window.

The campaign's initial access occurred February 28, 2026, when TeamPCP used an autonomous bot to steal a GitHub Personal Access Token belonging to a maintainer of Aqua Security's Trivy project — one of the most widely deployed open-source vulnerability scanners in the cloud-native ecosystem [1][9]. Trivy's design gives it privileged access by necessity: in a typical CI/CD environment, it reads container images, scans filesystem layers, and processes Kubernetes resource manifests, accumulating credentials and secrets in its operating environment as a matter of normal function. Compromising the tool rather than the secrets it processes amplifies the attack's reach: a single credential theft can yield access to every environment in which the tool runs with credential access — a far larger attack surface than targeting any individual application.

The compromise was not detected for three weeks. On March 19–20, 2026, Aqua Security disclosed the intrusion and assigned CVE-2026-33634 (CVSS 9.4, Critical), while simultaneously the CanisterWorm component began propagating autonomously [1][9][12]. CanisterWorm scanned the stolen credential store for npm publishing tokens, enumerated the corresponding packages, incremented patch version numbers, and republished 66 or more infected npm packages [7][8] — an operation that, based on prior documented CanisterWorm deployments, likely completed within seconds of the initial compromise [7]. This infection rate illustrates both the velocity of automated supply chain attacks and the structural risk of reusing credentials across package ecosystems.

The subsequent week saw TeamPCP expand its footprint systematically. On March 21–22, malicious Docker Hub images for Trivy (versions 0.69.5 and 0.69.6) were published to expand the infection surface beyond npm [1]. On March 23, Checkmarx's `kics-github-action` and `ast-github-action` — GitHub Actions broadly adopted for infrastructure-as-code security scanning — were compromised alongside multiple OpenVSX IDE extensions [13]. On March 24, LiteLLM versions 1.82.7 and 1.82.8 appeared on PyPI and remained available for approximately five hours before removal [22]. The WAV-steganography payload targeting the Telnyx Python SDK followed on March 27 [3][4].

Arctic Wolf, ReversingLabs, and Kaspersky independently assess attribution confidence across all sub-campaigns as high, citing reuse of the same RSA-4096 public key, consistent naming of the exfiltration archive as `tpcp.tar.gz`, and a shared command-and-control domain pattern of typosquatting legitimate project names [1][2][11]. By March 25, the FBI Cyber Division had issued an advisory, and reporting confirmed that TeamPCP had reportedly pivoted to active extortion, reportedly in collaboration with LAPSUS$ based on the threat actor's own announcements [14], while processing approximately 300 GB of compressed stolen credentials [8][13].

Security Analysis

The WAV Container Technique

The audio steganography component of the campaign, deployed in the compromised Telnyx PyPI package (versions 4.87.1 and 4.87.2), is technically distinctive and warrants careful characterization. The technique is not classical least-significant-bit (LSB) audio steganography, in which payload bits are embedded invisibly within the amplitude values of audio samples. Instead, TeamPCP constructed files that are structurally valid WAV containers — possessing a legitimate RIFF/WAVE header that passes format validation — but whose audio frame data consists entirely of encoded payload rather than audio samples [3][4].

The delivery mechanism, injected into `telnyx/_client.py`, executes on package import. It contacts a hardcoded C2 address at `83.142.209.203:8080` and retrieves a file with a `.wav` extension and a MIME type consistent with audio content. On Linux and macOS systems, the delivered file is `ringtone.wav`; on Windows, `hangup.wav` — a naming choice that aligns contextually with Telnyx's voice communications use case and would not raise immediate suspicion in a communications-platform development environment [3][4].

Decoding the payload requires only Python standard library components: the `wave` module reads the audio frame data, `base64.b64decode()` processes the result, and the first eight bytes of the decoded output serve as the XOR cipher key for decrypting the remainder. The final step executes the decrypted binary or shell script in memory. No third-party libraries with anomalous import patterns are required. On Windows, the payload drops a file named `msbuild.exe` into the Windows Startup folder for persistence; on Linux and macOS, the credential harvester runs without establishing persistence and then self-destructs [4].

The evasion rationale is layered. Audio MIME types traverse network content-type filters that block `application/octet-stream` or executable MIME types. The RIFF/WAVE header causes file-type scanners to classify the file as audio. The payload is encrypted until the final XOR step executes in memory. The Telnyx contextual cover — a communications SDK legitimately associated with audio functionality — provides an additional layer of social plausibility. The technique does not require kernel-level capabilities, anti-debugging logic, or any dependency that would trigger behavioral detection rules written for conventional malware loaders [3][4][11].

The affected Telnyx package had approximately 740,000 to over one million monthly downloads, depending on measurement window and source [3][15], and the malicious versions were available for approximately six hours before removal.

LiteLLM and the `.pth` Persistence Mechanism

The LiteLLM compromise employed a distinct and particularly aggressive persistence technique. Python's `.pth` file mechanism — a standard feature of the Python packaging ecosystem that allows packages to specify paths to be added to `sys.path` at interpreter startup — can be abused to execute arbitrary code on every Python process initialization, not only when the malicious package is explicitly imported [5][6].

TeamPCP's injected `litellm_init.pth` file exploited this mechanism, ensuring that the credential stealer ran regardless of whether an infected host ever executed code that directly referenced LiteLLM. Any Python process on the compromised host — including CI/CD runners, automated scripts, and development tools with no LiteLLM dependency — would silently execute the stealer. The malicious `_proxy_server.py` modification complemented this, targeting SSH private keys, AWS and GCP credential files, Kubernetes service account tokens, environment files containing API keys, and any `.env` file in the working directory [5][6].

Exfiltration used a typosquatting domain, `models.litellm[.]cloud`, crafted to appear as a legitimate LiteLLM infrastructure endpoint. With approximately 95 million monthly downloads [5][21], the actual installation count during the approximately five-hour availability window is difficult to determine from the download metric alone [5][6][22].

CanisterWorm and Blockchain C2

The self-propagating CanisterWorm component employed a previously undocumented infrastructure choice for supply chain attack C2: use of the Internet Computer Protocol (ICP) blockchain as a command-and-control channel [7][8]. ICP canisters — smart contract instances on the ICP decentralized network — can host arbitrary computation and store state in a manner that is not addressable through the conventional registrar, hosting provider, or CDN takedown mechanisms that threat intelligence operations typically employ. The decentralized governance model of ICP substantially increases C2 resilience compared to conventional domain-based infrastructure [7][8].

The CanisterWorm C2 architecture creates a specific challenge for incident response operations reliant on conventional takedown mechanisms. The CanisterWorm C2 address, `tdtqy-oyaaa-aaaae-af2dq-cai.raw.icp0.io`, is a content-addressed identifier on a decentralized network [7]. Conventional C2 disruption via domain suspension does not apply to ICP canister endpoints; defenders cannot take down the C2 channel through registrar or hosting contacts. Effective response relies instead

on host-level containment — detecting and disabling `pgmon.service`, reimaging affected systems — along with behavioral network monitoring for traffic to the specific canister identifier and egress filtering for the `icp0.io` namespace in environments where Web3 access is not a business requirement.

The worm's propagation loop operated in four stages: a Node.js postinstall hook decoded and wrote a Python backdoor to `~/.local/share/pgmon/service.py`; a systemd service named `pgmon.service`, disguised as a PostgreSQL monitoring daemon with `Restart=always`, ensured persistent re-execution; the deployed service polled the ICP canister for updated instructions; and a credential-scanning module searched for npm publishing tokens to enumerate and re-infect additional packages [7][8]. The pace of re-publication at peak activity suggests a high degree of automation with minimal operator intervention required after initial deployment.

Structural Risk: Tool-Centric Targeting

The campaign's target selection is consistent with a deliberate strategy of privileged-tool compromise rather than application-layer attack — a pattern that is either a feature of deliberate planning or an emergent property of targeting high-value, widely-deployed tooling. Trivy, KICS, and the Checkmarx GitHub Actions all occupy positions in the security scanning layer of the software development lifecycle, where they are granted broad read access to the same secrets and configurations they are designed to assess. LiteLLM, by virtue of serving as an API routing layer between applications and LLM providers, processes authentication tokens for every major AI platform in a single credential surface. The Telnyx SDK connects a developer's environment to a communications platform that may hold telephony credentials, number provisioning tokens, and call data.

In each case, the compromise of the tool — rather than a vulnerability in the application it serves — grants access to the full credential surface that the tool was trusted to process. This pattern represents a structural challenge for organizations whose security controls are calibrated to detect anomalous behavior in application code rather than anomalous behavior by security infrastructure itself.

Recommendations

Immediate Actions

Organizations that installed any of the following should treat all secrets in affected environments as compromised and initiate credential rotation immediately:

- Trivy container images v0.69.5 or v0.69.6 from Docker Hub (any platform)
- LiteLLM PyPI versions 1.82.7 or 1.82.8
- Telnyx PyPI versions 4.87.1 or 4.87.2
- Any of the 66+ CanisterWorm-infected npm packages identified in Arctic Wolf and ReversingLabs advisories [1][2]
- `kics-github-action` or `ast-github-action` used in CI/CD pipelines between March 23–25, 2026 [13]

Credential rotation should encompass SSH private keys, cloud provider credentials (AWS, GCP, Azure), Kubernetes service account tokens, API keys present in `.env` files, npm and PyPI publishing tokens, and GitHub Personal Access Tokens. Given the `.pth` persistence mechanism in the LiteLLM compromise, any Python environment that installed affected versions should be treated as fully compromised — not only the LiteLLM dependency context.

Short-Term Mitigations

Organizations should audit their dependency trees for indirect exposure. A direct dependency on any listed package is not required for exposure: any CI/CD pipeline that ran on an infected runner, or any development environment that shared a Python interpreter with an infected package install, may have been affected by the `.pth` propagation mechanism.

Network logging should be reviewed for outbound connections to `83.142.209.203:8080`, the domain `models.litellm[.]cloud`, and ICP canister endpoints in the `icp0.io` namespace during the relevant windows [5][7]. The presence of a `pgmon.service` systemd unit on any Linux host warrants immediate investigation, as does any file named `litellm_init.pth` in the site-packages directories of Python installations [5][7].

Package integrity verification should be standardized across build environments. For organizations already pinned to specific prior versions via cryptographic hashes in `requirements.txt` and `package-lock.json` files, hash verification would have prevented automatic installation of the malicious package versions during the availability window. Tools including pip-audit, Socket.dev, and Phylum can be integrated into CI/CD pipelines to detect newly published packages with anomalous behavioral patterns before installation [3][15].

Strategic Considerations

The use of ICP blockchain infrastructure for C2 requires a revision of conventional supply chain incident response assumptions. Organizations should review whether their network egress filtering would detect or block traffic to decentralized Web3 network endpoints, including ICP canisters, Ethereum node RPC interfaces, and IPFS gateways, which are not addressed by domain-based blocklist controls. Where blocking is operationally feasible without unacceptable collateral impact, restricting outbound access to these categories represents a prudent defense-in-depth measure for environments that do not have a business requirement for direct Web3 network access.

The audio container technique highlights a broader detection gap: content inspection rules that rely on MIME type or file extension classification can be bypassed by structurally valid containers whose payload is encrypted at rest. Network security monitoring should be augmented with behavioral heuristics that flag unexpected download of audio files from infrastructure not associated with legitimate audio content delivery. Static analysis integrations should be reviewed to ensure that Python standard library usage patterns — particularly `wave`, `base64`, and in-memory execution via `exec()` — are evaluated in the context of import-time hooks and postinstall scripts.

Finally, the LiteLLM compromise illustrates the systemic risk of the AI developer toolchain as an attack surface. Security architecture reviews of AI toolchain components should apply the same rigor as reviews of identity providers and secrets management systems. Although these components are not purpose-built for credential custody, they process API keys for multiple cloud AI providers, route sensitive inference data, and often run with broad filesystem permissions — creating a credential exposure surface comparable in practice to that of dedicated secrets systems.

CSA Resource Alignment

The TeamPCP campaign engages directly with multiple frameworks maintained by the Cloud Security Alliance. The MAESTRO framework's Layer 4 (Deployment and Infrastructure) addresses the class of risks realized here: compromised execution environments, container image poisoning, and the integrity of the runtime in which AI workloads execute [17]. MAESTRO Layer 3 (Agent Frameworks) is implicated specifically by the LiteLLM compromise, which targeted the orchestration layer through which AI agent applications route all inference traffic. The campaign reinforces MAESTRO's design premise that each layer must be treated as a potential adversary-controlled surface, and it surfaces a gap in current framework guidance: existing MAESTRO controls do not explicitly address the scenario in which a security tool in the deployment layer itself becomes an adversarial surface — a scenario that warrants dedicated treatment in future framework updates.

The CSA AI Controls Matrix (AICM) v1.0 maps directly to several control domains compromised in this campaign. The Supply Chain Security domain addresses dependency integrity and package verification controls that would have limited exposure to CanisterWorm and the PyPI-distributed payloads. The Runtime Security domain covers the persistent `.pth` execution mechanism and systemd service installation that constituted the campaign's host-level persistence. The Secrets and Credential Management domain is the primary impact category: the campaign's purpose was credential exfiltration, and AICM controls in this domain — including rotation policy, scope minimization, and just-in-time provisioning — directly mitigate the blast radius of the compromise [18].

The CSA Cloud Controls Matrix (CCM) v4.0 Infrastructure and Virtualization Security (IVS) and Supply Chain Management, Transparency, and Accountability (STA) domains apply to the container image poisoning dimension of the campaign: the Trivy Docker Hub images represent a supply chain integrity failure at the container image distribution layer, where cryptographic image signing (IVS-04) and third-party software verification (STA-05) would have enabled earlier detection [19].

The CSA's Zero Trust guidance is particularly relevant to the structural risk pattern identified in this campaign. Zero Trust architectures that treat security tooling as an untrusted workload — applying least-privilege network policies and runtime behavioral monitoring to scanning infrastructure itself — would have constrained the blast radius of the Trivy and Checkmarx compromises, even assuming no advance detection of the initial credential theft. The assumption that security tools occupy a trusted operational layer is inconsistent with Zero Trust principles and is directly exploited by tool-centric targeting campaigns of this type [20].

References

[1] Arctic Wolf, "TeamPCP Supply Chain Attack Campaign Targets Trivy, Checkmarx KICS, and LiteLLM," Arctic Wolf Blog, March 2026. https://arcticwolf.com/resources/blog/teampcp-supply-chain-attack-campaign-targets-trivy-checkmarx-kics-and-litellm-potential-downstream-impact-to-additional-projects/

[2] ReversingLabs, "Inside the TeamPCP Cascading Supply Chain Attack," ReversingLabs Blog, March 2026. https://www.reversinglabs.com/blog/teampcp-supply-chain-attack-spreads

[3] BleepingComputer, "Backdoored Telnyx PyPI Package Pushes Malware Hidden in WAV Audio," BleepingComputer, March 2026. https://www.bleepingcomputer.com/news/security/backdoored-telnyx-pypi-package-pushes-malware-hidden-in-wav-audio/

[4] OSSPrey, "New .WAV of TeamPCP Malware," OSSPrey Blog, March 2026. https://ossprey.com/blog/telnyx-pypi-malware-wav/

[5] Sonatype, "Compromised LiteLLM PyPI Package Delivers Multi-Stage Credential Stealer," Sonatype Blog, March 2026. https://www.sonatype.com/blog/compromised-litellm-pypi-package-delivers-multi-stage-credential-stealer

[6] Upwind, "LiteLLM Supply Chain Attack RCE Analysis," Upwind Blog, March 2026. https://www.upwind.io/feed/litellm-pypi-supply-chain-attack-malicious-release

[7] MrCloudBook, "TeamPCP CVE-2026-33634 Full Analysis," MrCloudBook Blog, March 2026. https://mrcloudbook.com/teampcp-supply-chain-attack-telnyx-canisterworm-full-analysis-cve-2026-33634/

[8] SANS Institute, "When the Security Scanner Became the Weapon: Inside the TeamPCP Supply Chain Campaign," SANS Blog, March 2026. https://www.sans.org/blog/when-security-scanner-became-weapon-inside-teampcp-supply-chain-campaign

[9] Aqua Security, "Trivy Supply Chain Attack: Ongoing Investigation and Continued Remediation," Aqua Security Blog, March 2026. https://www.aquasec.com/blog/trivy-supply-chain-attack-what-you-need-to-know/

[10] GitHub Security Advisory GHSA-69fq-xp46-6x23, "Trivy Supply Chain Compromise," GitHub, March 2026. https://github.com/aquasecurity/trivy/security/advisories/GHSA-69fq-xp46-6x23

[11] Kaspersky, "Trojanization of Trivy, Checkmarx, and LiteLLM," Kaspersky Blog, March 2026. https://www.kaspersky.com/blog/critical-supply-chain-attack-trivy-litellm-checkmarx-teampcp/55510/

[12] Tenable, "CVE-2026-33634 Detail," Tenable CVE Database, March 2026. https://www.tenable.com/cve/CVE-2026-33634

[13] Sysdig, "TeamPCP Expands Supply Chain Compromise: Spreads from Trivy to Checkmarx GitHub Actions," Sysdig Blog, March 2026. https://www.sysdig.com/blog/teampcp-expands-supply-chain-compromise-spreads-from-trivy-to-checkmarx-github-actions

[14] The Hacker News, "TeamPCP Pushes Malicious Telnyx Versions to PyPI," The Hacker News, March 2026. https://thehackernews.com/2026/03/teampcp-pushes-malicious-telnyx.html

[15] SafeDep, "WAV Steganography and Credential Theft: The telnyx PyPI Compromise," SafeDep Blog, March 2026. https://safedep.io/malicious-telnyx-pypi-compromise/

[16] Help Net Security, "LiteLLM PyPI Packages Compromised in TeamPCP Campaign," Help Net Security, March 2026. https://www.helpnetsecurity.com/2026/03/25/teampcp-supply-chain-attacks/

[17] Cloud Security Alliance, "MAESTRO: Agentic AI Threat Modeling Framework," CSA AI Safety Initiative, 2025. https://cloudsecurityalliance.org/research/working-groups/ai-safety/

[18] Cloud Security Alliance, "AI Controls Matrix (AICM) v1.0," CSA AI Safety Initiative, 2026. https://cloudsecurityalliance.org/research/topics/artificial-intelligence/

[19] Cloud Security Alliance, "Cloud Controls Matrix v4.0," CSA Research, 2021. https://cloudsecurityalliance.org/research/cloud-controls-matrix/

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

[21] Endor Labs, "TeamPCP Isn't Done: Threat Actor Behind Trivy and KICS Compromises Now Hits LiteLLM's 95 Million Monthly Downloads on PyPI," Endor Labs Blog, March 2026. https://www.endorlabs.com/learn/teampcp-supply-chain-attack-litellm-pypi

[22] Xygeni, "LiteLLM Supply Chain Attack: How TeamPCP Backdoored AI Infrastructure," Xygeni Blog, March 2026. https://xygeni.io/blog/litellm-supply-chain-attack-how-teampcp-backdoored-ai-infrastructure/

