INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-21 18:37 UTC

TeamPCP Supply Chain Attack: Malicious Telnyx PyPI SDK Versions Deliver Steganographic Credential-Stealing Malware

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0194
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Telnyx Python SDK (PyPI versions 4.87.1 and 4.87.2); Python environments on Linux, macOS, and Windows; Kubernetes clusters
Published	2026-04-21T03:14:56
Discovery Source	Rss

Executive Summary

Threat actor TeamPCP compromised the official Telnyx Python SDK on PyPI, publishing malicious versions 4.87.1 and 4.87.2 that steal SSH keys, cloud credentials, and Kubernetes secrets from any environment that installed them. Any organization using the Telnyx Python SDK in development pipelines, CI/CD systems, or production infrastructure may have had credentials silently exfiltrated. The Telnyx SDK is widely used in the Python developer community, making the potential scope significant, and because the malware conceals its payload inside audio files, standard file-scanning tools likely missed it.

Technical Analysis

TeamPCP, a supply chain threat actor previously tied to the LiteLLM PyPI compromise, injected malicious code into Telnyx Python SDK versions 4.87.1 and 4.87.2 on PyPI. The payload is steganographically embedded within WAV audio files (MITRE T1027.003, Steganography), bypassing signature-based detection that inspects Python source files or binaries directly. Upon package import, the malware executes automatically (T1059.006, Python) and harvests: SSH private keys (T1552.004), cloud provider authentication tokens for AWS, GCP, and Azure (T1528, Steal Application Access Token), cryptocurrency wallet data, and Kubernetes cluster secrets (T1613, Container and Resource Discovery). Exfiltration occurs over a command-and-control channel (T1041, Exfiltration Over C2 Channel). Persistence mechanisms consistent with T1543.001 have been observed. The attack targets Linux, macOS, and Windows Python environments, as well as Kubernetes clusters. Relevant

CWEs: CWE-506 (Embedded Malicious Code), CWE-494 (Download of Code Without Integrity Check), CWE-522 (Insufficiently Protected Credentials), CWE-312 (Cleartext Storage of Sensitive Information). No CVE has been assigned. Telnyx has confirmed the compromise via official security notice and GitHub issue #235. Safe versions are those predating 4.87.1 or postdating 4.87.2, verify against Telnyx's official advisory before upgrading. Technical claims are corroborated across T1 and T2 sources: SANS ISC Diary (T1), Telnyx official notice (vendor-official), GitHub issue (official repository), and security publications including BleepingComputer and OX Security.

Action Checklist

1. Step 1: Containment. Immediately identify any environment, CI/CD pipeline, container image, or developer workstation that installed telnyx versions 4.87.1 or 4.87.2. Run 'pip show telnyx' or 'pip list' across all Python environments. Isolate affected systems from the network before proceeding. Check container images built after the approximate publication date of the malicious versions.
2. Step 2: Detection. Search package manager logs, CI/CD build logs, and container image manifests for 'telnyx==4.87.1' or 'telnyx==4.87.2'. Inspect requirements.txt, pyproject.toml, Pipfile.lock, and Poetry lock files across all repositories. Review outbound network connections from Python processes for anomalous destinations, particularly at import time. Look for unexpected WAV files in package directories under site-packages/telnyx. Query SIEM for DNS lookups or HTTP connections originating from Python interpreter processes to unknown external hosts.
3. Step 3: Eradication. Uninstall the malicious package immediately: 'pip uninstall telnyx'. Do not upgrade in place without first consulting Telnyx's official security notice (published March 2026 on their website and in their GitHub repository) to confirm which version is safe. Rebuild any container images that included the affected versions from a clean base. Rotate all credentials that may have been present in affected environments: SSH private keys, AWS/GCP/Azure tokens and IAM credentials, Kubernetes service account tokens and kubeconfig files, cryptocurrency wallet keys.
4. Step 4: Recovery. After credential rotation, audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Monitor) for unauthorized API calls or resource access that may have occurred post-compromise. Review Kubernetes audit logs for unauthorized cluster access. Verify that CI/CD pipelines have been rebuilt from clean images. Confirm that reinstalled telnyx package hash matches the known-good version per Telnyx's advisory before returning systems to production.
5. Step 5: Post-Incident. Review your software supply chain controls: enforce hash-pinning or digest-pinning for all PyPI dependencies, not just major packages. Implement a private package mirror or artifact proxy (e.g., Artifactory, AWS CodeArtifact) with allowlisting. Add automated scanning for steganographic payloads in package assets; standard SAST and AV tools missed this attack. Map this incident to CIS Control 2 (Inventory and Control of Software Assets) and NIST SP 800-161 (Supply Chain Risk Management) gaps. Brief the development team on TeamPCP's pattern of targeting widely-used Python SDKs.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO, legal counsel, and external IR retainer if CloudTrail, GCP Audit Logs, or Kubernetes audit logs confirm any API calls, resource access, or cluster activity using credentials that were present in an environment where telnyx 4.87.1 or 4.87.2 was installed — confirmed credential use post-compromise triggers breach notification assessment under applicable regulations (GDPR Article 33, US state breach notification statutes, HIPAA §164.412 if PHI was accessible to the compromised environment).
Recovery Notes	After credential rotation and pipeline rebuild, maintain enhanced monitoring of all cloud provider accounts and Kubernetes clusters for a minimum of 30 days — TeamPCP's operational pattern of targeting developer environments and CI/CD pipelines means stolen credentials may be staged for delayed use rather than immediate exploitation. Verify integrity of any infrastructure-as-code (Terraform, Pulumi, Helm charts) stored in repositories accessible from compromised developer workstations, as these may have been exfiltrated and could be used to reconstruct environment topology for follow-on attacks. Confirm that all reinstalled telnyx package hashes are pinned in requirements files and that CI/CD pipeline jobs fail hard on hash mismatch before returning any affected pipeline to active use.
Forensic Artifacts	Malicious telnyx package directory at $(python3 -c 'import site; print(site.getsitepackages()[0])')/telnyx/ — contains the steganographic WAV file(s) embedding the credential-stealing payload; must be preserved before uninstall as it is the primary malware artifact for this TeamPCP campaign | pip install logs and CI/CD build logs containing 'telnyx==4.87.1' or 'telnyx==4.87.2' install events — establish the precise timestamp and environment scope of initial compromise, required to bound the credential exfiltration window | Outbound network connection records from Python interpreter processes (auditd SYSCALL connect records on Linux, Sysmon EventID 3 on Windows, macOS unified log network events) capturing the C2 or exfiltration endpoint contacted by the steganographic payload at import time | Cloud provider credential usage logs (AWS CloudTrail, GCP Audit Logs, Azure Monitor Activity Log) scoped from telnyx install timestamp through credential rotation — documents whether TeamPCP operationalized any stolen AWS/GCP/Azure tokens before rotation | Kubernetes API server audit log and kubeconfig files from affected developer workstations — establishes which cluster endpoints and namespaces were accessible via stolen kubeconfig credentials, defining the full Kubernetes blast radius of this TeamPCP intrusion

Per-Action IR Details

Step 1: Containment — Immediately identify any environment, CI/CD pipeline, container image, or developer workstation that installed telnyx versions 4.87.1 or 4.87.2. Run 'pip show telnyx' or 'pip list' across all Python environments. Isolate affected systems from the network before proceeding. Check container images built after the approximate publication date of the malicious versions.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Run the following one-liner across all reachable hosts via Ansible, SSH loop, or manual execution: `python3 -c "import pkg_resources; v=pkg_resources.get_distribution('telnyx').version; print(f'{v}') if v in ['4.87.1','4.87.2'] else None" 2>/dev/null`. For container images, enumerate with `docker images --format '{{.Repository}}:{{.Tag}}'` and inspect each with `docker run --rm pip show telnyx`. For Kubernetes, use `kubectl get pods -A -o json | jq '.items[].spec.containers[].image'` to enumerate all running images, then `docker inspect` or `crane config` to check installed packages. A 2-person team can script this as a parallel SSH job using `pssh` or `pdsh` against the full host inventory.

Evidence: Before isolating, capture a full snapshot of the affected Python environment: `pip freeze > /tmp/pip_freeze_$(hostname)_$(date +%s).txt`. Capture running process tree (`ps auxf` on Linux/macOS, `Get-Process` on Windows) to document any Python processes active at time of discovery — TeamPCP's payload executes at import time, so a running Python process importing telnyx may indicate live exfiltration in progress. Preserve `/proc//net/tcp` and `/proc//net/tcp6` on Linux for any Python PIDs to capture open socket state before network isolation. On Kubernetes, capture `kubectl describe pod ` and `kubectl get events` output before terminating pods.

Step 2: Detection — Search package manager logs, CI/CD build logs, and container image manifests for 'telnyx==4.87.1' or 'telnyx==4.87.2'. Inspect requirements.txt, pyproject.toml, Pipfile.lock, and Poetry lock files across all repositories. Review outbound network connections from Python processes for anomalous destinations, particularly at import time. Look for unexpected WAV files in package directories under site-packages/telnyx. Query SIEM for DNS lookups or HTTP connections originating from Python interpreter processes to unknown external hosts.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For WAV-based steganographic payload discovery, locate the telnyx site-packages directory with `python3 -c "import telnyx; import os; print(os.path.dirname(telnyx.__file__))"` then run `find -name '*.wav' -o -name '*.png' -o -name '*.jpg'` — presence of any binary media files in a pure-Python SDK package directory is anomalous and warrants immediate analysis. Extract and inspect with `strings ` or `binwalk ` (install via `apt install binwalk`) to reveal embedded payloads. For network detection without a SIEM, enable Sysmon (Windows) with EventID 3 (Network Connection) filtered on `python.exe` or `python3` as the initiating process, or use `auditd` on Linux with a rule: `-a always,exit -F arch=b64 -S connect -F exe=/usr/bin/python3 -k telnyx_net`. On macOS, use `sudo fs_usage python3 | grep -E 'connect|send'`. For CI/CD log grep: `grep -rE 'telnyx==(4\.87\.1|4\.87\.2)' --include='*.txt' --include='*.toml' --include='*.lock' .` across all cloned repos.

Evidence: Preserve the intact malicious package directory at `$(python3 -c 'import site; print(site.getsitepackages()[0])')/telnyx/` as a forensic image before uninstalling — this is the primary artifact containing the steganographic WAV file and obfuscated loader code. Capture a hash of all files: `find -type f -exec sha256sum {} \; > telnyx_package_hashes.txt`. Extract outbound network connection history from: `/var/log/syslog` or `journalctl -u ` for Linux daemons running the SDK, GitHub Actions/GitLab CI job logs for any `pip install` steps, and AWS CloudTrail `CreateNetworkInterface` or `RunInstances` events if the package executed in Lambda or EC2. On Kubernetes, collect `kubectl logs --previous` for any pods that ran the affected image.

Step 3: Eradication — Uninstall the malicious package immediately: 'pip uninstall telnyx'. Do not upgrade in place without first consulting Telnyx's official security notice (https://telnyx.com/resources/telnyx-python-sdk-supply-chain-security-notice-march-2026) to confirm which version is safe. Rebuild any container images that included the affected versions from a clean base. Rotate all credentials that may have been present in affected environments: SSH private keys, AWS/GCP/Azure tokens and IAM credentials, Kubernetes service account tokens and kubeconfig files, cryptocurrency wallet keys.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST IA-5 (Authenticator Management), NIST CM-2 (Baseline Configuration), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 5.2 (Use Unique Passwords)

Compensating: After uninstalling, verify removal and re-install a known-good version with hash verification: `pip uninstall -y telnyx && pip install telnyx== --require-hashes --hash=sha256:`. Obtain the correct hash from Telnyx's official advisory or by cross-referencing PyPI's JSON API: `curl -s https://pypi.org/pypi/telnyx//json | python3 -m json.tool | grep sha256`. For SSH key rotation on Linux: `find ~/.ssh /home/*/.ssh /root/.ssh -name 'id_*' -not -name

'*.pub' 2>/dev/null` to enumerate all private keys that were accessible in the compromised environment. For AWS, enumerate exposed keys with `aws iam list-access-keys --user-name ` and rotate via `aws iam create-access-key` followed by `aws iam delete-access-key`. For Kubernetes, rotate service account tokens: `kubectl delete secret -n ` to force regeneration. Document every rotated credential with timestamp for audit trail.

Evidence: Before uninstalling, forensically preserve the malicious package: `cp -r $(python3 -c 'import site; print(site.getsitepackages()[0])')/telnyx /tmp/forensic_telnyx_$(date +%s)` and generate a manifest: `find /tmp/forensic_telnyx_* -type f -exec sha256sum {} \;`. Capture the pip install log that originally pulled the malicious version from `~/.pip/pip.log` or `$(pip config get global.log)` if configured. Preserve any `.pyc` compiled cache files under `__pycache__` within the telnyx package directory — these may contain deobfuscated bytecode of the steganographic loader recoverable via `uncompyle6` or `decompile3`. On CI/CD systems (GitHub Actions, Jenkins, GitLab CI), download and archive the full build log artifact for the job that performed the `pip install telnyx` step before log retention expires.

Step 4: Recovery — After credential rotation, audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Monitor) for unauthorized API calls or resource access that may have occurred post-compromise. Review Kubernetes audit logs for unauthorized cluster access. Verify that CI/CD pipelines have been rebuilt from clean images. Confirm that reinstalled telnyx package hash matches the known-good version per Telnyx's advisory before returning systems to production.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-11 (Audit Record Retention), NIST CP-10 (System Recovery and Reconstitution), CIS 8.2 (Collect Audit Logs)

Compensating: For AWS CloudTrail analysis without a SIEM, use CloudTrail Lake or the following CLI query scoped to the compromise window: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue= --start-time --end-time `. Focus on `AssumeRole`, `GetSecretValue`, `DescribeInstances`, `CreateUser`, and `PutBucketPolicy` event names — these represent post-compromise lateral movement and persistence actions TeamPCP's stolen credentials would enable. For Kubernetes audit log review without a SIEM, use: `kubectl get events -A --field-selector reason=Forbidden` and parse the API server audit log at `/var/log/kubernetes/audit.log` with `jq` filtering on `verb: ["create","delete","patch"]` and `user.username` values that do not match known service accounts. For package hash verification: `pip show telnyx | grep -i location` then `sha256sum $(pip show telnyx | grep Location | awk '{print $2}')/telnyx-*.dist-info/RECORD`.

Evidence: Collect and preserve cloud provider audit logs covering the full window from the PyPI publication date of 4.87.1 through credential rotation completion — this is the definitive exfiltration and unauthorized access window for this TeamPCP campaign. For AWS: export CloudTrail logs from S3 to local storage with `aws s3 sync s3://////CloudTrail/ ./cloudtrail_forensic/`. For Kubernetes: export audit logs from the API server and any managed Kubernetes audit logging service (e.g., GKE Audit Logs, EKS CloudTrail integration). Preserve the `~/.kube/config` file from any affected developer workstation as evidence of which clusters were accessible to the stolen kubeconfig — TeamPCP's credential theft specifically targeted kubeconfig files, and the cluster endpoints listed there define the full blast radius.

Step 5: Post-Incident — Review your software supply chain controls: enforce hash-pinning or digest-pinning for all PyPI dependencies, not just major packages. Implement a private package mirror or artifact proxy (e.g., Artifactory, AWS CodeArtifact) with allowlisting. Add automated scanning for steganographic payloads in package assets — standard SAST and AV missed this attack. Map this incident to CIS Control 2 (Inventory and Control of Software Assets) and NIST SP 800-161 (Supply Chain Risk Management) gaps. Brief the development team on TeamPCP's pattern of targeting widely-used Python SDKs.

NIST Phase: Post Incident

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

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Protection), NIST IR-8 (Incident Response Plan), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability

Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Implement pip hash-pinning immediately using `pip-compile --generate-hashes` from the `pip-tools` package to regenerate all `requirements.txt` files with per-package SHA-256 digests — this is free and directly mitigates the mechanism TeamPCP used (unsigned PyPI package replacement). For steganographic payload detection in CI/CD pipelines without commercial tooling, write a YARA rule targeting binary media file types (WAV, PNG, JPEG) present in installed Python package directories and run it as a pipeline step: `yara -r media_in_packages.yar $(python3 -c 'import site; print(site.getsitepackages()[0])')`. Use `truffleHog` (free, open source) to scan all repositories and CI/CD environment variables for newly committed secrets that may have been exfiltrated and reused by TeamPCP. Add an `osquery` scheduled query to all developer workstations: `SELECT name, version, location FROM python_packages WHERE name='telnyx'` with alerting on version values matching 4.87.1 or 4.87.2. For the development team brief, reference TeamPCP's confirmed prior campaigns targeting PyPI-distributed Python SDKs to establish the pattern of targeting transitive dependencies in developer toolchains.

Evidence: Produce a post-incident artifact package containing: (1) the forensically preserved malicious telnyx package directory with all file hashes, (2) the full timeline of installation events across all affected environments reconstructed from pip logs and CI/CD build logs, (3) the complete list of credentials confirmed in-scope for rotation with rotation timestamps, and (4) cloud provider audit log exports covering the compromise window. This package supports both internal lessons-learned and any required regulatory breach notification documentation. Preserve all evidence for a minimum of 12 months per NIST AU-11 (Audit Record Retention) requirements, or longer if regulatory obligations (SOC 2, PCI-DSS, HIPAA) apply to the affected environments.

Detection Guidance

Primary detection: query all Python environment package lists for telnyx==4.87.1 or telnyx==4.87.2 using 'pip list --format=freeze | grep telnyx'. Scan lock files (requirements.txt, Pipfile.lock, poetry.lock, pip-compile output) across all repositories for these pinned versions. In CI/CD systems, search build logs for installation of either version. Secondary detection: look for WAV or audio files within the telnyx package directory under site-packages; legitimate SDK packages do not contain audio files. Their presence is a strong indicator of compromise. Network-based detection: alert on outbound connections from Python interpreter processes (python, python3) to external IP addresses not associated with Telnyx's legitimate API endpoints, particularly at application import time rather than during active API calls. Behavioral indicator: unexpected reads of ~/.ssh/ directories, cloud credential files (~/.aws/credentials, ~/.config/gcloud/, ~/.azure/), or kubeconfig files by a process in the Python SDK import chain. SIEM query approach: correlate process name containing 'python' with file access events touching SSH key paths and outbound network events within the same short time window.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`telnyx==4.87.1 (PyPI)`	Malicious PyPI package version — treat any installation as compromised; obtain specific file hashes from Telnyx's official security notice and OX Security blog	HIGH
HASH	`telnyx==4.87.2 (PyPI)`	Malicious PyPI package version — treat any installation as compromised; obtain specific file hashes from Telnyx's official security notice and OX Security blog	HIGH

Type	Value	Context	Confidence
URL	`https://github.com/team-telnyx/telnyx-python/issues/235`	Telnyx GitHub issue confirming compromise — review for any additional IOCs published by the community	HIGH
URL	`https://isc.sans.edu/diary/32838`	SANS ISC Diary entry 32838 — TeamPCP Supply Chain Campaign Update 002; may contain network IOCs not included in this item	HIGH

Framework Mappings

MITRE-ATTACK

- **T1195.002** — Compromise Software Supply Chain
- **T1041** — Exfiltration Over C2 Channel
- **T1543.001** — Launch Agent
- **T1552.001** — Credentials In Files
- **T1528** — Steal Application Access Token
- **T1610** — Deploy Container
- **T1059.006** — Python
- **T1027.003** — Steganography
- **T1613** — Container and Resource Discovery
- **T1552.004** — Private Keys

NIST-800-53R5

- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **IA-5** — Authenticator Management
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **5.2** — Use Unique Passwords
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1195.002	Compromise Software Supply Chain	Initial-Access
T1041	Exfiltration Over C2 Channel	Exfiltration
T1543.001	Launch Agent	Persistence
T1552.001	Credentials In Files	Credential-Access
T1528	Steal Application Access Token	Credential-Access
T1610	Deploy Container	Defense-Evasion
T1059.006	Python	Execution
T1027.003	Steganography	Defense-Evasion
T1613	Container and Resource Discovery	Discovery
T1552.004	Private Keys	Credential-Access

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/backdoored-telnyx-py...	T3
Telnyx Python SDK: Supply Chain Security Notice	https://telnyx.com/resources/telnyx-python-sdk-supply-chain-securit...	T3
[SECURITY] PyPI versions 4.87.1 and 4.87.2 are compromised	https://github.com/team-telnyx/telnyx-python/issues/235	T3
Telnyx Malware: TeamPCP Strikes Again - OX Security	https://www.ox.security/blog/telnyx-malware-teampcp-strikes-again-f...	T3
TeamPCP Supply Chain Campaign: Update 002 - Telnyx PyPI ...	https://isc.sans.edu/diary/32838	T1

Generated 2026-04-21 18:37 UTC by TJS Security Command Center