INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-27 18:49 UTC

elementary-data PyPI Supply Chain Attack: GitHub Actions Script Injection Delivers Signed Infostealer via Legitimate Release Pipeline

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0226
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	elementary-data PyPI package v0.23.3 and prior; GitHub Actions CI/CD pipeline; GitHub Container Registry (ghcr.io/elementary-data/elementary); dbt ecosystem integrations
Published	2026-04-27T11:17:37
Discovery Source	Rss

Executive Summary

An attacker hijacked the CI/CD pipeline of elementary-data, a widely used Python analytics package with 1.1 million monthly downloads, and published a backdoored release (v0.23.3) that was cryptographically signed and visually identical to a legitimate update. The malicious package targets cloud credentials (AWS, GCP, Azure), SSH private keys, CI/CD secrets, and cryptocurrency wallet files. Any organization that installed v0.23.3 or pulled the associated Docker image must treat all secrets accessible in those environments as fully compromised.

Technical Analysis

The attacker exploited a GitHub Actions script injection vulnerability in the elementary-data project's CI/CD workflow, enabling execution of attacker-controlled commands within the trusted release pipeline without compromising any developer account. The resulting release (v0.23.3) was published simultaneously to PyPI and GitHub Container Registry (ghcr.io/elementary-data/elementary) with a valid cryptographic signature, bypassing publisher verification and package signing controls entirely. The injected payload functions as an infostealer targeting cloud provider credential files (AWS ~/.aws/credentials, GCP application default credentials, Azure CLI tokens), SSH private keys, CI/CD environment secrets, and cryptocurrency wallet files. Relevant CWEs: CWE-77 (Command Injection), CWE-78 (OS Command Injection via untrusted input), CWE-732 (Incorrect Permission Assignment), CWE-346 (Origin Validation Error). MITRE ATT&CK techniques: T1195.001 (Compromise Software Supply Chain), T1528 (Steal Application Access Token), T1552.001 (Credentials in

Files), T1552.004 (Private Keys), T1555 (Credentials from Password Stores), T1059.004 (Unix Shell), T1650 (Acquire Access). No CVE assigned as of source publication date. Affected: elementary-data PyPI package v0.23.3 and prior builds sharing the compromised pipeline window; associated Docker image on GHCR. Source: StepSecurity (T3), BleepingComputer (T3).

Action Checklist

1. Containment: Immediately identify all systems, pipelines, and CI/CD environments where elementary-data v0.23.3 was installed via pip or where the ghcr.io/elementary-data/elementary Docker image was pulled. Isolate those environments from production systems and revoke all secrets accessible within them pending rotation.
2. Detection: Query package installation logs, pip audit outputs, and container registries for elementary-data==0.23.3 or the associated Docker image digest. In cloud environments, audit AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for anomalous API calls or credential use originating from CI/CD pipelines that ran in the affected window. Check for unexpected outbound connections from build runners.
3. Eradication: Upgrade elementary-data to the latest verified clean release via pip (confirm version is post-0.23.3 and sourced from the official PyPI package page). Remove and re-pull any Docker images sourced from ghcr.io/elementary-data/elementary pulled during the compromise window. Re-evaluate GitHub Actions workflow files in any project using elementary-data for signs of persisted script injection payloads.
4. Recovery: Rotate all secrets that were accessible in any environment where v0.23.3 executed: AWS IAM access keys, GCP service account keys, Azure service principal credentials, SSH private keys, and any CI/CD secrets (GitHub Actions secrets, environment variables). Revalidate pipeline integrity by reviewing GitHub Actions workflow logs for the compromise window. Monitor cloud provider logs for 30 days post-rotation for residual unauthorized access.
5. Post-Incident: Review all GitHub Actions workflows across your organization for script injection patterns: untrusted input (PR titles, branch names, issue bodies) interpolated directly into run steps. Implement workflow hardening per StepSecurity HARDEN-RUNNER or equivalent. Evaluate adoption of package pinning by hash (not version tag) for PyPI dependencies in CI/CD. Consider software composition analysis (SCA) tools that flag newly published versions before pipeline installation.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal, and external IR retainer immediately if AWS CloudTrail, GCP Audit Logs, or Azure Activity Logs show API calls authenticated with credentials accessible during the v0.23.3 execution window originating from IPs outside known organizational infrastructure, or if any CI/CD secret with production environment scope was present in an affected workflow — both conditions likely trigger breach notification obligations under GDPR, SOC 2, or applicable state privacy law depending on what data those credentials could access.

Recovery Notes	After rotating all credentials, establish a 30-day enhanced monitoring baseline in your cloud provider logs specifically watching for the old compromised key IDs, old GCP service account key fingerprints, and old SSH public key fingerprints — attacker use of credentials after rotation confirms exfiltration and active abuse, not just collection. Revalidate that all GitHub Actions workflows consuming elementary-data are pinned to a verified clean version by hash and that no workflow-level `GITHUB_TOKEN` with `write` permissions was exposed during the compromise window, as token abuse could enable repository tampering beyond credential theft. Confirm pipeline integrity by performing a clean re-run of affected CI/CD workflows in an isolated environment and comparing build artifacts against known-good checksums before restoring production pipeline operations.
Forensic Artifacts	pip cache directory artifacts: `~/.cache/pip/wheels/` (Linux/macOS) or `%LOCALAPPDATA%\pip\Cache\wheels\` (Windows) — the cached elementary-data-0.23.3 wheel file is the primary malware sample; hash it with SHA256 and preserve before any eradication steps | GitHub Actions runner diagnostic logs: located at `/_diag/Runner_*.log` and `Worker_*.log` on self-hosted runners, or downloadable via `GET /repos/{owner}/{repo}/actions/runs/{run_id}/logs` for GitHub-hosted runners — these logs capture the exact pip install invocation, environment variable exposure, and any subprocess activity spawned by the malicious package during the workflow run | Cloud provider credential usage telemetry: AWS CloudTrail events for `GetCallerIdentity`, `AssumeRole`, `ListBuckets`, and `GetSecretValue` with `sourceIPAddress` matching GitHub Actions IP ranges (meta.githubusercontent.com published list) during the compromise window — the infostealer would have called STS first to enumerate permissions before targeting high-value resources | Filesystem timeline artifacts on build runners: `/tmp/` and `/var/tmp/` directories for staged credential files (the infostealer targets `~/.aws/credentials`, `~/.ssh/id_rsa*`, `~/.config/gcloud/application_default_credentials.json`, and cryptocurrency wallet files under `~/Library/Application Support/` or `~/.config/`); capture with `find /tmp /var/tmp ~/.aws ~/.ssh ~/.config -newer -ls` before eradication | Network connection records: DNS query logs or VPC flow logs from build runner subnets showing outbound connections to non-PyPI, non-GitHub domains initiated within the process tree of the `pip install elementary-data` or the subsequent workflow steps — the C2 exfiltration channel for this infostealer would appear as a short-lived HTTPS or DNS-over-HTTPS connection to an attacker-controlled domain immediately following credential file access, distinguishable from normal build traffic by destination reputation and timing correlation with package installation

Per-Action IR Details

Containment — Immediately identify all systems, pipelines, and CI/CD environments where elementary-data v0.23.3 was installed via pip or where the ghcr.io/elementary-data/elementary Docker image was pulled. Isolate those environments from production systems and revoke all secrets accessible within them pending rotation.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-3 (Malicious Code Protection), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Run `pip list --format=json | python3 -c "import sys,json; pkgs=json.load(sys.stdin); print([p for p in pkgs if p['name']=='elementary-data' and p['version']=='0.23.3'])"` on all build hosts. For Docker: `docker images --format '{{.Repository}}:{{.Tag}} {{.ID}}' | grep ghcr.io/elementary-data/elementary` and cross-reference pulled image digests against the compromise window. Use osquery `SELECT * FROM python_packages WHERE name='elementary-data' AND version='0.23.3';` for fleet-wide enumeration. Immediately block outbound egress from affected runners using host firewall rules (`iptables -I OUTPUT -j DROP` or Windows Firewall via `netsh advfirewall`).

Evidence: BEFORE isolating: snapshot the pip package cache (`~/.cache/pip/` or `C:\Users\\AppData\Local\pip\Cache\`) to preserve the downloaded wheel for malware analysis. Capture Docker layer digests: `docker inspect ghcr.io/elementary-data/elementary --format='{{.Id}} {{.RepoDigests}}'`. Export GitHub Actions runner logs from `./_work/_temp/` or the runner's `_diag/` folder. Image the build runner's `/tmp/` and `/var/tmp/` directories — infostealer payloads frequently stage exfiltrated credential files there prior to C2 transmission. Record all active network connections at time of isolation: `ss -tulnp` (Linux) or `netstat -ano` (Windows).

Detection — Query package installation logs, pip audit outputs, and container registries for elementary-data==0.23.3 or the associated Docker image digest. In cloud environments, audit AWS CloudTrail, GCP Audit Logs, and Azure Activity Logs for anomalous API calls or credential use originating from CI/CD pipelines that ran in the affected window. Check for unexpected outbound connections from build runners.

NIST Phase: Detection Analysis

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

Controls: NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST SI-4 (System Monitoring), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: AWS: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetSecretValue --start-time --end-time ` and filter for `userAgent` containing `python-boto3` or `python-requests` from runner IPs. GCP: `gcloud logging read 'protoPayload.methodName="iam.serviceAccounts.signBlob" OR protoPayload.methodName="storage.objects.list"' --freshness=7d`. Azure: query Activity Log for `Microsoft.Authorization/roleAssignments/write` or `Microsoft.KeyVault/vaults/secrets/read` from pipeline service principals. For SSH key theft: check `~/.ssh/known_hosts` modification timestamps and `auth.log` / `/var/log/secure` for unexpected SSH session initiations from build runner IPs. Use Wireshark or `tcpdump -i any -w capture.pcap 'not port 443 and not port 80'` on runners to catch non-standard C2 channels.

Evidence: AWS CloudTrail: filter `eventSource=s3.amazonaws.com` and `sts.amazonaws.com` for `AssumeRole`, `GetCallerIdentity`, and `ListBuckets` events originating from GitHub Actions runner IP ranges during the compromise window — these are the first API calls an infostealer would make to enumerate accessible resources. GCP: look for `storage.buckets.list` and `secretmanager.versions.access` calls. GitHub Actions workflow run logs (downloadable via API: `GET /repos/{owner}/{repo}/actions/runs/{run_id}/logs`) — search for unexpected `curl`, `wget`, or `python -c` invocations not present in the workflow YAML. Container registry pull logs from ghcr.io (accessible via GitHub Packages audit log in org settings). Host-level: `/proc//net/tcp` or equivalent for active connections at time of execution.

Eradication — Upgrade elementary-data to the latest verified clean release via pip (confirm version is post-0.23.3 and sourced from the official PyPI package page). Remove and re-pull any Docker images sourced from ghcr.io/elementary-data/elementary pulled during the compromise window. Re-evaluate GitHub Actions workflow files in any project using elementary-data for signs of persisted script injection payloads.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Verify clean PyPI package integrity: `pip download elementary-data --no-deps -d /tmp/pkg_verify && sha256sum /tmp/pkg_verify/*.whl` — compare against the SHA256 hash published on the official PyPI package page for the clean release. For Docker: `docker rmi $(docker images --format '{{.Repository}}:{{.Tag}} {{.ID}}' | grep ghcr.io/elementary-data/elementary | awk '{print $2}')` then re-pull with explicit digest pinning: `docker pull ghcr.io/elementary-data/elementary@sha256:`. For GitHub Actions workflow audit, run: `grep -rn '\${{.*github.event.pull_request\|github.event.issue\|github.head_ref' .github/workflows/` to surface untrusted context variables interpolated into `run:` steps — this is the exact injection pattern exploited in this campaign. Use YARA rule targeting `import base64` + `subprocess` + credential path strings (`/.aws/credentials`, `/.ssh/id_rsa`) within `.py` files in site-packages.

Evidence: Before removing the malicious package: extract and hash `elementary-data-0.23.3.dist-info/RECORD` to document the installed file manifest — this establishes the forensic ground truth of what the backdoor touched. Preserve a copy of the malicious wheel from pip cache for malware analysis. Capture `git log --all --oneline` and `git diff` for any `.github/workflows/*.yml` files modified during the compromise window — persisted injection payloads would appear as added `run:` steps referencing external URLs or base64-encoded commands. Check for cron jobs or systemd timers created by the infostealer: `crontab -l`, `/etc/cron.d/`, `systemctl list-timers`.

Recovery — Rotate all secrets that were accessible in any environment where v0.23.3 executed: AWS IAM access keys, GCP service account keys, Azure service principal credentials, SSH private keys, and any CI/CD secrets (GitHub Actions secrets, environment variables). Revalidate pipeline integrity by reviewing GitHub Actions workflow logs for the compromise window. Monitor cloud provider logs for 30 days post-rotation for residual unauthorized access.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST IA-5 (Authenticator Management), NIST AU-11 (Audit Record Retention), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: AWS: `aws iam list-access-keys --user-name ` then `aws iam delete-access-key` and `aws iam create-access-key` — document old key IDs for CloudTrail correlation during the 30-day monitoring window. GCP: `gcloud iam service-accounts keys list --iam-account=@.iam.gserviceaccount.com` then delete and regenerate. GitHub Actions secrets: navigate to repo/org Settings → Secrets → rotate each value; note that GitHub does not log secret access, so treat all secrets present in any workflow that called elementary-data as compromised. SSH keys: `ssh-keygen -R ` to clear known_hosts entries, then regenerate key pairs and push new public keys via your provisioning tool. Post-rotation, set a CloudWatch alarm or GCP Log-based alert on the old AWS key IDs / old GCP key fingerprints appearing in any log — attacker use of rotated credentials confirms exfiltration occurred.

Evidence: Before rotating AWS keys: run `aws iam generate-credential-report && aws iam get-credential-report` to capture last-used timestamps for all access keys — this establishes whether the infostealer actually used the exfiltrated credentials or only exfiltrated them. Preserve the full CloudTrail event history for the compromise window (download to S3 with object lock enabled per NIST AU-9 (Protection of Audit Information)). Document all GitHub Actions secret names present in affected workflows — these names become your rotation checklist and the evidence record for any regulatory breach notification.

Post-Incident — Review all GitHub Actions workflows across your organization for script injection patterns: untrusted input (PR titles, branch names, issue bodies) interpolated directly into run steps. Implement workflow hardening per StepSecurity HARDEN-RUNNER or equivalent. Evaluate adoption of package pinning by hash (not version tag) for PyPI dependencies in CI/CD. Consider software composition analysis (SCA) tools that flag newly published versions before pipeline installation.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SI-10 (Information Input Validation), NIST CM-3 (Configuration Change Control), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Script injection audit: `grep -rn '\${{\s*github\.event\.' .github/workflows/ | grep -v '#'` across all org repos — pipe results to a CSV for triage. Pin PyPI dependencies by hash in requirements files: `pip-compile --generate-hashes requirements.in` using pip-tools (free, OSS). For SCA without enterprise budget, use `pip-audit` (free, maintained by PyPA): `pip-audit -r requirements.txt --output=json` — integrate into CI as a pre-install gate that fails the pipeline on newly published package versions not yet reviewed. Implement Sigma rule for GitHub Actions anomaly detection: alert on workflow runs where a `pip install` step is followed within 60 seconds by outbound DNS queries to non-GitHub, non-PyPI domains (detectable via runner host DNS logs or VPC flow logs if runners are self-hosted). StepSecurity HARDEN-RUNNER is free for public repos and provides egress filtering and step auditing.

Evidence: Lessons-learned artifacts to preserve: the complete list of affected workflow run IDs and their SHA-pinned commit hashes (establishes the exact code that executed with the malicious package); the pip dependency tree from each affected environment (`pip freeze > freeze_evidence_.txt`) to identify transitive exposure; a timeline mapping elementary-data v0.23.3 installation timestamps against cloud API call spikes in CloudTrail/GCP Audit Logs — this correlation is the primary evidence for determining whether credential exfiltration was followed by active exploitation, which drives regulatory breach notification decisions.

Detection Guidance

Search package manifests, requirements.txt files, pip freeze outputs, and lock files for elementary-data==0.23.3. Query container image pull logs for ghcr.io/elementary-data/elementary images pulled during the compromise window. In GitHub Actions logs, look for unexpected outbound network connections or process executions during steps that installed or invoked elementary-data. In cloud provider logs: AWS CloudTrail, look for GetCallerIdentity, ListBuckets, or credential enumeration calls from CI/CD runner IPs or assumed roles that ran the affected pipeline; GCP Cloud Audit Logs, review for service account key exports or unusual API calls from build infrastructure; Azure Monitor, review for token acquisition events from CI/CD sources. Check SSH authorized_keys and known_hosts for unexpected additions post-pipeline execution. Behavioral indicator: outbound connections to unfamiliar IPs from build runners immediately following elementary-data installation or execution steps.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`Not published in available reporting`	Malicious elementary-data v0.23.3 package hash — check StepSecurity advisory for confirmed digest values	LOW
URL	`https://pypi.org/project/elementary-data/0.23.3/`	Compromised PyPI release page — do not install; reference only for version identification	HIGH
DOMAIN	`ghcr.io/elementary-data/elementary`	GitHub Container Registry image associated with compromised release — do not pull images from this path without verifying digest against a confirmed clean release	HIGH

Framework Mappings

MITRE-ATTACK

- **T1588.006** — Vulnerabilities
- **T1528** — Steal Application Access Token
- **T1078.001** — Default Accounts
- **T1555** — Credentials from Password Stores
- **T1195.001** — Compromise Software Dependencies and Development Tools

- **T1552.001** — Credentials In Files
- **T1650** — Acquire Access
- **T1552.004** — Private Keys
- **T1059.004** — Unix Shell

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-3** — Access Enforcement
- **AC-6** — Least Privilege
- **SI-10** — Information Input Validation
- **AT-2** — Literacy Training and Awareness
- **SR-2** — Supply Chain Risk Management Plan
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A03:2021** — Injection

CIS-V8

- **3.3** — Configure Data Access Control Lists
- **2.5** — Allowlist Authorized Software
- **16.10** — Apply Secure Design Principles in Application Architectures
- **6.3** — Require MFA for Externally-Exposed Applications
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **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.8.8** — Management of technical vulnerabilities
- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain
- **A.8.24** — Use of cryptography
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1588.006	Vulnerabilities	Resource-Development
T1528	Steal Application Access Token	Credential-Access
T1078.001	Default Accounts	Defense-Evasion
T1555	Credentials from Password Stores	Credential-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1650	Acquire Access	Resource-Development
T1552.004	Private Keys	Credential-Access
T1059.004	Unix Shell	Execution

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/pypi-package-with-11...	T3
elementary-data Compromised on PyPI and GHCR: Forged Release ...	https://www.stepsecurity.io/blog/elementary-data-compromised-on-pyp...	T3
PyPI package with 1.1M monthly downloads hacked to push ...	https://www.bleepingcomputer.com/news/security/pypi-package-with-11...	T3
Chainguard customers safe from elementary-data compromise	https://www.chainguard.dev/unchained/chainguard-customers-safe-from...	T3
Varun Sharma's Post - LinkedIn	https://www.linkedin.com/posts/varunsharma07_elementary-data-0233-i...	T3

Generated 2026-04-27 18:49 UTC by TJS Security Command Center