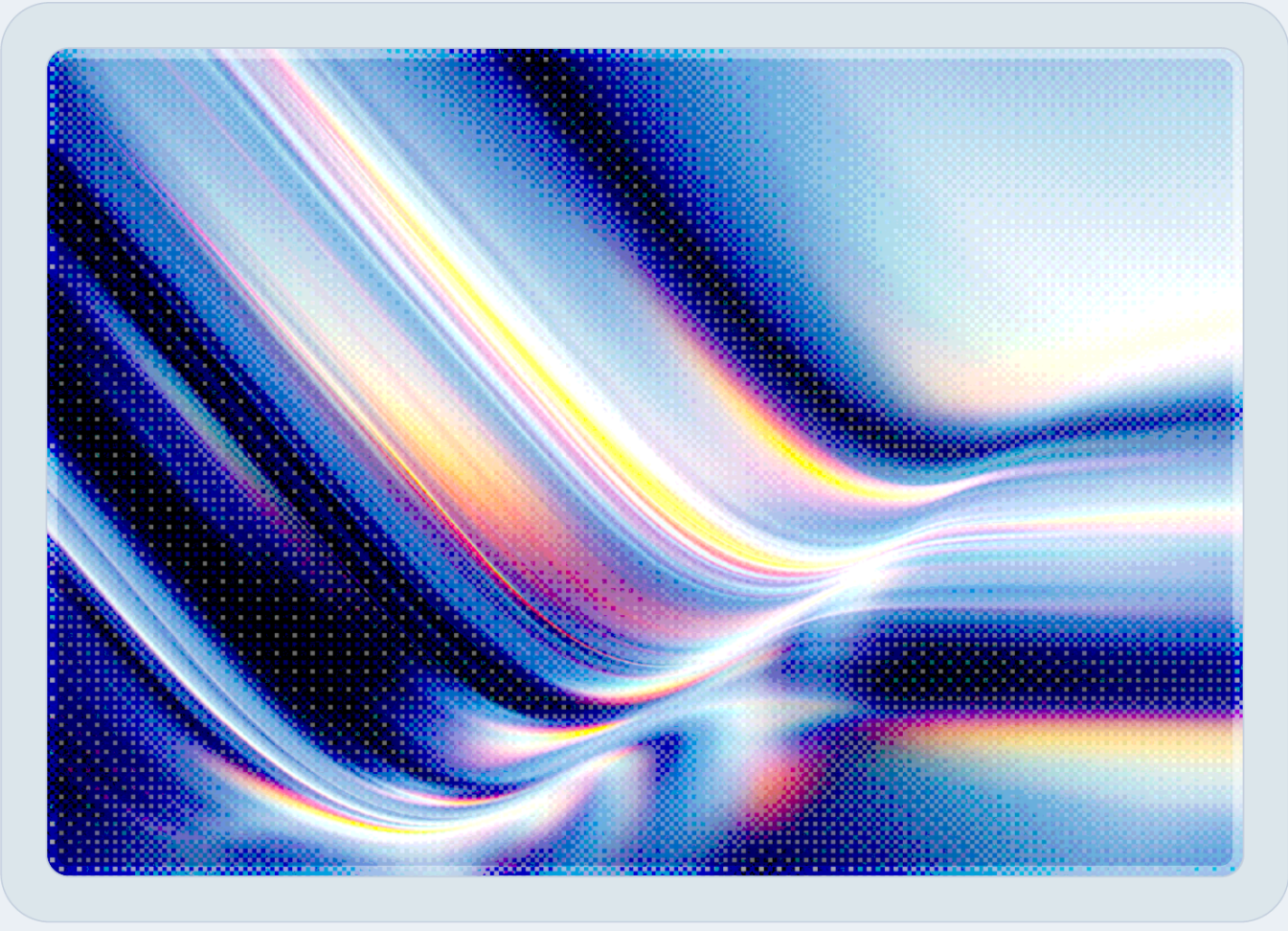
CSAI Foundation | Cloud Security Alliance

elementary-data PyPI Compromise: Cloud Credential Theft via CI/CD Hijack

GitHub Actions Script Injection Backdoors dbt Data Observability Package with 1.1M Monthly Downloads

2026-04-29

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

On April 24–25, 2026, attackers exploited a GitHub Actions script injection vulnerability in the elementary-data repository to forge a signed, legitimate-appearing release of version 0.23.3 and distribute it to PyPI and the GitHub Container Registry [1][2]. Elementary-data is a dbt-native data observability CLI used heavily by data and analytics engineers to monitor pipeline quality across cloud data warehouses [3]; the package logged over 1.1 million monthly downloads at the time of the compromise [1][4]. The compromised version embedded a Python path configuration file (`elementary.pth`) that executed a credential-harvesting infostealer automatically at every Python interpreter startup, before any user code ran [1][5].

The credentials targeted reflect the full data engineering ecosystem. The infostealer harvested AWS IAM credentials — including an IMDSv2 role-privilege lookup to capture instance role credentials in cloud-hosted environments — alongside GCP application default credentials, Azure service principal files, dbt warehouse profiles, Kubernetes service account tokens, Docker registry credentials, SSH private keys, `.env` files, cryptocurrency wallet configurations, and developer tool secrets including `.npmrc`, `.pypirc`, and HashiCorp Vault tokens [1][2]. Exfiltrated data was transmitted to an attacker-controlled command-and-control domain; organizations should treat any machine that ran `elementary-data==0.23.3` as compromised and rotate all cloud credentials accessible to processes running in that Python environment [1].

Version 0.23.4, released by the Elementary team on April 25, 2026, contains no malicious code [4]. The compromised 0.23.3 release has been removed from PyPI and from the GitHub Container Registry [2] [4]. This incident follows the March 2026 LiteLLM PyPI supply chain compromise by approximately thirty days and reinforces a concerning threat pattern: adversaries appear to have identified the high-trust CI/CD pipelines of data engineering and AI tooling as high-value targets — three major compromises in sixty days, each reaching cloud credentials these tools require to function, suggest this ecosystem merits elevated threat model attention regardless of whether the incidents are coordinated.

Background

Elementary-data is the CLI companion to the dbt-data-reliability package, together forming a dbt-native data observability platform used by data and analytics engineers to detect anomalies, monitor schema changes, track pipeline freshness, and deliver quality alerts to Slack and Microsoft Teams [3][6]. The tool occupies a privileged position in a typical data engineering environment: it connects directly to cloud data warehouses — Snowflake, BigQuery, Redshift, Databricks — using credentials stored in dbt profiles, and it often runs in automated contexts such as Airflow DAGs, CI/CD pipelines, and scheduled container workloads where the credentials are machine-managed rather than human-entered [6]. That combination of legitimate cloud access, widespread deployment, and automated execution makes it an attractive vector for credential theft.

The elementary-data GitHub repository uses a conventional open-source release workflow. Maintainers publish releases by creating and pushing tags, at which point GitHub Actions workflows build the Python package, push it to PyPI, and build and tag a multi-architecture Docker container image uploaded to the GitHub Container Registry at `ghcr.io/elementary-data/elementary` [2]. This workflow is intentionally automated for maintainer efficiency, but it also means that any actor who gains sufficient GitHub token permissions — whether a compromised maintainer account, a misconfigured workflow, or a script injection in a triggered action — can publish a forged release that carries the same provenance signals as a legitimate one.

The Attacker's Entry Point: A Malicious PR Comment

At 22:10 UTC on April 24, 2026, a GitHub account created just two days earlier — identified by StepSecurity researchers as `realtungtungtungsahur` — posted a crafted comment on pull request #2147 in the elementary-data repository [2]. The comment was not a meaningful code review; it was a payload disguised as text, exploiting a known class of vulnerability in GitHub Actions workflows: script injection via unsanitized event context values.

The vulnerable workflow, `.github/workflows/update_pylon_issue.yml`, was configured to trigger on `issue_comment` and `pull_request_review_comment` events. Its `run:` block directly interpolated the value of `${{ github.event.comment.body }}` — the verbatim content of the poster's comment — into a shell script before bash parsing occurred [2]. This is a documented anti-pattern that GitHub's own hardening guides warn against, because the comment body is attacker-controlled and is substituted into the script source before shell quoting rules apply. When the

workflow triggered on the attacker's comment, the embedded payload executed with the runner's `GITHUB_TOKEN` in scope. That token carried `contents: write` permissions, sufficient to create commits, push tags, and dispatch additional workflows [2].

Security Analysis

Forging a Signed Release via CI/CD Pipeline Abuse

The value of GitHub Actions script injection in a supply chain attack is precisely that it allows an attacker to move a malicious artifact through the victim organization's own release infrastructure. Rather than compromising PyPI credentials directly — which would require stealing a maintainer's API token — the attacker abused the repository's existing publish-to-PyPI workflow. Releases built and pushed by the repository's own CI/CD pipeline inherit all the legitimacy signals that downstream consumers rely upon: they appear in the project's official release history, they carry the maintainer's package signing identity, and they are tagged and version-pinned in the same way as clean releases [1][2].

StepSecurity's analysis confirmed that the attacker used the stolen `GITHUB_TOKEN` to inject a malicious file into the source tree, commit it, and create the tag `v0.23.3`, which in turn triggered the repository's automated publish workflow [2]. The same workflow that data engineers had configured in their `pip install` and `requirements.txt` files then faithfully packaged and distributed the backdoored code. An identical flow pushed a multi-architecture container image to GHCR tagged both `0.23.3` and `latest`, extending exposure to any organization pulling the Docker image for containerized dbt environments [2].

The .pth File: Execution Before User Code

The malicious payload was delivered not as a modified script or monkey-patched module but as a Python path configuration file, `elementary.pth`, placed in the package's `site-packages` directory [1][5]. This delivery mechanism exploits a little-examined feature of the Python runtime: `.pth` files are processed by `site.py` — Python's startup module — during interpreter initialization, before the user's own code begins executing. Any line in a `.pth` file that begins with the token `import` is evaluated as Python code at startup [1]. This means the infostealer activates on every Python invocation in an affected environment, including subprocess calls, scheduled jobs, and server processes, not only direct invocations of the `edr` elementary CLI.

The persistence and stealth properties of the `.pth` execution technique are noteworthy. A developer inspecting their installed packages with `pip show elementary-data` or examining the `edr` entry point would find nothing obviously unusual; the malicious execution path lies in a file type that most developers have never directly inspected. Many standard antivirus and EDR tools focused on executable file types may miss `.pth`-based triggers, since these files are plain text and the malicious logic is loaded dynamically at runtime [1] — though endpoint agents with Python interpreter telemetry may detect the anomalous module load.

Credential Exfiltration Scope and Targeting

The infostealer's targeting profile reveals deliberate knowledge of the data engineering tool stack. Beyond the standard credential sweep targeting SSH keys, cloud provider credential files, and Kubernetes secrets, the malware explicitly targeted `~/.dbt/profiles.yml`, which stores the warehouse passwords and OAuth tokens used by dbt and elementary-data to connect to Snowflake, BigQuery, Redshift, and other platforms [1]. In environments where least-privilege principles have not been applied to dbt service accounts, these profiles may contain credentials with broad data warehouse access, potentially including production datasets. The IMDSv2 role lookup extends credential capture to cloud-hosted runtime environments — Airflow workers, dbt Cloud runners, and analytics notebook servers — where instance roles may carry substantial data warehouse and storage permissions.

Exfiltrated data was sent to the domain `igotnofriendsonlineorirl-imgonnakmslmao.skyhanni.cloud`, the attacker's command-and-control infrastructure [1]. Organizations investigating potential exposure should search proxy, DNS, and network flow logs for connections to this domain originating from any host with a Python environment. The infostealer also dropped a marker file — `/tmp/.trinny-security-update` on macOS and Linux, and `%TEMP%\.trinny-security-update` on Windows — which can be used as an indicator of compromise for endpoint forensics [4].

The malicious release was live on PyPI for approximately eight to ten hours [1]: a community member identified as `crisperik` opened an issue on the project's GitHub repository on April 25, 2026, alerting the maintainers, who moved quickly to remove version 0.23.3 and publish the clean 0.23.4 [4]. Nevertheless, given that the package averaged roughly 280,000 downloads per week [1], the exposure window was sufficient to affect a meaningful number of installations, particularly in automated pipeline environments where `pip install` is triggered on a schedule or as part of CI runs [4].

Recommendations

Immediate Actions

Organizations that deployed `elementary-data==0.23.3` or pulled the `ghcr.io/elementary-data/elementary:0.23.3` or `ghcr.io/elementary-data/elementary:latest` container image between approximately 22:10 UTC April 24 and approximately 08:00 UTC April 25, 2026 — primary sources estimate the window at roughly eight to ten hours [1][2] — should treat those environments as compromised and initiate credential rotation immediately. The rotation scope should encompass dbt warehouse credentials, AWS IAM access keys and any instance roles accessible from affected hosts, GCP service account keys, Azure service principal credentials, Kubernetes service account tokens, Docker registry credentials, SSH private keys, and any secrets stored in `.env` files accessible to the Python process [1]. Rotating credentials without investigating access logs is insufficient: organizations should audit AWS CloudTrail, GCP Audit Logs, and Azure Monitor for unusual API activity — particularly calls to IAM, storage, or data warehouse APIs from unexpected sources or timeframes — beginning from the time of first exposure.

Endpoint responders should scan for the marker files `/tmp/.trinny-security-update` (macOS/Linux) and `%TEMP%\.trinny-security-update` (Windows) as indicators of infostealer execution, and should inspect all `.pth` files in affected Python virtual environments for unauthorized content [4]. Network defenders should query proxy and DNS logs for connections to `igotnofriendsonlineorirl-imgonnakmslmao.skyhanni.cloud` [1].

Short-Term Mitigations

For the elementary-data package specifically, organizations should pin to `elementary-data==0.23.4` or later and verify the installed version with `pip show elementary-data` in every environment [4]. Organizations using containerized dbt environments should rebuild all images that pulled the `latest` tag during the exposure window and confirm they are running 0.23.4-based images. Package managers that cache or mirror PyPI internally should purge any cached copy of version 0.23.3.

This incident also demonstrated the value of factory-level detection in the software supply chain: Chainguard's build pipeline identified the malicious package before it was incorporated into any Chainguard artifact, providing a concrete example of how upstream verification controls can limit downstream exposure [10]. More broadly, this incident illustrates the risk of workflows that grant write-

level GitHub tokens to jobs triggered by external contributor events. Engineering teams should audit their GitHub Actions workflows for any `run:` blocks that interpolate untrusted event context values — particularly `github.event.comment.body`, `github.event.issue.title`, `github.event.pull_request.title`, and similar fields — and either remove the interpolation or use an intermediate environment variable assignment to break the injection path. GitHub's own Actions security hardening guide recommends treating all user-controlled context values as untrusted and never passing them directly into shell commands [2]. Workflow permissions should also be scoped to the minimum required: publish jobs should have `contents: read` unless a specific step requires write access, and PyPI publish credentials should be stored as environment secrets with the narrowest possible scope.

Strategic Considerations

Three notable PyPI supply chain compromises affecting this ecosystem — the March 2026 LiteLLM compromise, earlier incidents targeting MCP server packages, and now elementary-data — occurred within a sixty-day period, each reaching the cloud credentials these tools require to function. Whether these incidents represent coordinated attacks or independent opportunistic actors, the pattern warrants elevated attention: data engineering and AI tooling occupies a high-value position in organizational infrastructure, these tools are typically deployed with service account credentials that have broad data-plane permissions, and CI/CD pipeline hardening — particularly the prevention of script injection from external contributor input — has not been uniformly adopted across open-source data tooling projects, as this incident illustrates.

Organizations should treat their data and AI tooling pipeline as a high-sensitivity attack surface on par with their production application stack. Concrete measures include enforcing Sigstore or comparable artifact signing for all PyPI dependencies in production environments, deploying a software composition analysis tool that monitors for newly published versions of dependencies and flags installs of freshly released packages above a configurable version number, and implementing network egress filtering for data pipeline hosts that restricts outbound connections to known-good endpoints. Package pinning in `requirements.txt` and `pyproject.toml` with hash verification (`pip install --require-hashes`) prevents silent upgrades but must be combined with a process for reviewing and approving version bumps before they enter production. Container image immutability — pinning images by digest rather than tag — closes the vector that exposed organizations pulling `ghcr.io/elementary-data/elementary:latest` without realizing the tag had been reassigned to a backdoored build.

CSA Resource Alignment

This incident maps primarily to MAESTRO Layer 4 (Deployment and Infrastructure), addressing supply chain compromise of tooling that underpins cloud data infrastructure [8][11]. Organizations using elementary-data within AI or ML pipeline orchestration may also find MAESTRO Layer 3 (Agent Frameworks and Orchestration) applicable, as the same credential-access patterns apply when dbt and elementary-data feed data into agentic systems. MAESTRO's supply chain threat category within these layers addresses exactly the risk realized here: an adversary compromises a dependency in one layer to gain access to credentials and data flows across the broader agentic infrastructure stack. Organizations applying MAESTRO to their data platform threat models should enumerate all third-party packages with cloud credential access and assess whether each package's CI/CD pipeline would permit a forged release to reach production environments.

The CSA AI Controls Matrix (AICM) provides control guidance applicable at several points in this attack chain. Supply chain integrity controls under the AICM address verification of third-party component provenance, including artifact signing and dependency pinning requirements. Identity and access management controls speak to the principle of least privilege for service accounts used by data pipeline tools — a principle whose violation amplified the blast radius of this compromise by providing the infostealer access to credentials with data warehouse, storage, and IAM scope. The AICM's guidance on secrets management recommends that cloud credentials accessed by automated tools should be stored in secrets managers with short-lived, auto-rotating tokens wherever possible, rather than as long-lived static keys in credential files on disk.

CSA's Software Transparency: Securing the Digital Supply Chain publication provides a complementary framework for the organizational controls dimension, including SBOM generation for data pipeline dependencies and third-party risk assessment criteria for open-source tooling used in production data flows [9]. STAR registry participants operating AI or data engineering platforms should document their supply chain verification controls and artifact signing policies as part of their registry submission, providing a mechanism for downstream customers to assess vendor posture on exactly this risk.

References

[1] Snyk. "Malicious Release of elementary-data PyPI Package Steals Cloud Credentials from Data Engineers." Snyk Blog, April 2026.

[2] StepSecurity. "elementary-data Compromised on PyPI and GHCR: Forged Release Pushed via GitHub Actions Script Injection." StepSecurity Blog, April 2026.

[3] Elementary. "Elementary OSS Introduction." Elementary Documentation, 2026.

[4] BleepingComputer. "PyPI package with 1.1M monthly downloads hacked to push infostealer." BleepingComputer, April 2026.

[5] The CyberSec Guru. "Elementary-Data PyPI Hack: 1.1M Users Targeted by Infostealer." The CyberSec Guru, April 2026.

[6] Elementary-Data. "Elementary — The dbt-native data observability solution." GitHub, 2026.

[7] CybersecurityNews. "Popular PyPI Package With 1 Million Monthly Downloads Hacked to Inject Malicious Scripts." CybersecurityNews, April 2026.

[8] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." CSA Blog, February 2025.

[9] Cloud Security Alliance. "Software Transparency: Securing the Digital Supply Chain." CSA Research, 2025.

[10] Chainguard. "Chainguard customers safe from elementary-data compromise." Chainguard Blog, April 2026.

[11] Cloud Security Alliance. "Applying MAESTRO to Real-World Agentic AI Threats." CSA Blog, February 2026.