INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-25 13:52 UTC

TeamPCP Shai-Hulud Wave 3: Checkmarx Distribution Infrastructure Compromised via Multi-Vector Supply Chain Attack

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0219
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Checkmarx KICS Docker images, Checkmarx AST GitHub Action, Checkmarx AST Results VS Code extension, CX Dev Assist VS Code extension, npm ecosystem (@bitwarden/cli impersonation package), AWS SSM, Azure Key Vault, Google Cloud Secret Manager, GitHub Actions CI/CD environments, VS Code, Docker Hub
Published	2026-04-24T21:40:33+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

TeamPCP, an active threat actor targeting developer security tooling, has compromised multiple Checkmarx products simultaneously, including Docker images, GitHub Actions workflows, VS Code extensions, and a spoofed npm package, in the third wave of its supply chain campaign. The attack targets the CI/CD pipelines of organizations that use Checkmarx DevSecOps tooling, meaning the very tools deployed to enforce security are being used as the entry point. Any organization running affected Checkmarx components in their software delivery pipeline should treat this as an active compromise until investigation confirms otherwise.

Technical Analysis

TeamPCP's Wave 3 supply chain attack (April 2026) is a multi-vector compromise affecting Checkmarx KICS Docker Hub images, the Checkmarx AST GitHub Action, the Checkmarx AST Results VS Code extension, and the CX Dev Assist VS Code extension. A spoofed npm package (@bitwarden/cli v2026.4.0) served as an additional vector, impersonating the legitimate Bitwarden CLI used in developer pipelines. All vectors share a unified C2 infrastructure at audit.checkmarx[.]cx, a typosquatted domain impersonating the legitimate vendor. Payload behavior maps to the following CWEs: CWE-798 (hardcoded credentials), CWE-506 (embedded malicious code), CWE-829 (inclusion of functionality from untrusted control sphere), CWE-312 (cleartext storage

of sensitive information), CWE-494 (download of code without integrity check). Once executed, the payload harvests credentials and secrets from AWS SSM Parameter Store, Azure Key Vault, Google Cloud Secret Manager, GitHub tokens, and local developer workstations (T1552.004, T1555, T1552.001). The worm component (T1195.001, T1195.002) then injects malicious GitHub Actions workflows into every repository the compromised identity has push access to (T1098, T1136.003), enabling lateral propagation across the victim's entire supply chain. Obfuscation techniques (T1027) and defense evasion via disabling security controls (T1562.001) are also documented. No CVE identifier has been issued at this time. Patch status: consult official Checkmarx advisories for confirmed clean artifact hashes and remediated versions.

Action Checklist

1. Step 1: Containment, Immediately suspend all CI/CD pipeline jobs that reference Checkmarx KICS Docker images, the Checkmarx AST GitHub Action, Checkmarx AST Results VS Code extension, or CX Dev Assist VS Code extension. Block outbound connections to audit.checkmarx[.]cx at perimeter and endpoint DNS. Revoke all GitHub tokens, AWS SSM access keys, Azure Key Vault credentials, and GCP Secret Manager service accounts used in affected pipelines pending rotation.
2. Step 2: Detection, Query CI/CD logs and GitHub Actions workflow history for any reference to audit.checkmarx[.]cx or @bitwarden/cli v2026.4.0. Search npm lock files (package-lock.json, yarn.lock) across all repositories for @bitwarden/cli pinned to version 2026.4.0. Review Docker pull history for KICS images pulled from Docker Hub after 2026-04-01. Audit GitHub Actions workflow files for newly injected steps or unexpected uses of secrets contexts. Check cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Audit Logs) for unexpected SSM, Key Vault, or Secret Manager reads originating from CI/CD service accounts.
3. Step 3: Eradication, Replace all affected Checkmarx artifacts with versions confirmed clean by Checkmarx's official advisory (verify artifact integrity hashes before re-deployment). Remove @bitwarden/cli v2026.4.0 from all dependency manifests and replace with the verified legitimate package from the official Bitwarden registry, confirming the source URL and hash. Audit every repository the compromised identity had push access to; revert any injected workflow files. Remove or disable VS Code extensions pending confirmed clean versions from Checkmarx.
4. Step 4: Recovery, Rotate all secrets that were accessible during the exposure window: GitHub personal access tokens and Actions secrets, AWS IAM credentials with SSM access, Azure service principals with Key Vault access, GCP service accounts with Secret Manager access. Validate artifact integrity checksums against Checkmarx-published hashes before re-enabling pipelines. Re-enable pipelines in a staged manner with enhanced logging. Monitor for anomalous repository commits, unexpected workflow additions, or unusual cloud API calls for a minimum of 30 days post-remediation.
5. Step 5: Post-Incident, This attack succeeded by compromising trusted security tooling, bypassing controls that rely on source trustworthiness. Implement artifact integrity verification (e.g., Sigstore/cosign for container images, npm provenance attestation) across all CI/CD pipelines. Enforce least-privilege for CI/CD service accounts, no pipeline identity should have push access to repositories beyond its defined scope. Add DNS monitoring for typosquatted vendor domains as a standing detection control. Review the MITRE ATT&CK Supply Chain Compromise techniques (T1195.001, T1195.002) against your current control coverage and identify gaps.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and relevant cloud provider security teams immediately if CloudTrail, Azure Monitor, or GCP Audit Logs confirm any 'GetSecretValue', 'GetParameter', or 'SecretManagerGetSecretValue' API calls from CI/CD service accounts during the exposure window, as confirmed secret exfiltration from SSM, Key Vault, or Secret Manager constitutes a credential breach that may trigger regulatory notification obligations (SOC 2, PCI-DSS, GDPR, or state breach notification laws depending on the secrets' scope) — also escalate if any repository receiving push commits from the compromised identity contains production deployment workflows, as this extends the blast radius beyond the CI/CD environment to production systems.
Recovery Notes	Before re-enabling any Checkmarx-dependent pipeline, independently verify the SHA256 hash of every replacement artifact (KICS Docker image digest, AST GitHub Action pinned commit SHA, VS Code extension VSIX hash) against values published in Checkmarx's official advisory — do not trust Docker Hub tags alone, as tags are mutable. Re-enable pipelines one at a time in non-production environments first, with enhanced logging capturing every outbound network connection from the runner and every secrets context access, and validate clean operation for at least 72 hours before promoting to production pipelines. Maintain heightened monitoring for anomalous GitHub workflow additions, unexpected cloud secret reads, and DNS lookups to checkmarx-adjacent domains for a minimum of 30 days, as TeamPCP's Shai-Hulud campaign history (Wave 1, Wave 2) demonstrates a pattern of re-entry through residual access after initial remediation.
Forensic Artifacts	GitHub Actions workflow run logs (downloadable via GitHub API as zip bundles) for all runs referencing checkmarx/ast-github-action or checkmarx/kics during the TeamPCP exposure window — these contain stdout output of any injected exfiltration step sending data to audit.checkmarx[.]cx, including any base64-encoded secret values captured by the malicious action | DNS query logs from CI/CD runner hosts for resolutions of 'audit.checkmarx.cx' — obtainable from systemd-resolved journals, Zeek dns.log, or perimeter DNS resolver query logs; the presence and timing of these queries establishes the exfiltration window and whether the C2 domain was successfully reached | AWS CloudTrail 'GetParameter' and 'GetSecretValue' events filtered by CI/CD IAM role ARNs during the exposure window — these records confirm whether SSM parameters or Secrets Manager values were read by the compromised pipeline identity and are the primary evidence for cloud credential exfiltration scope | npm cache tarballs of @bitwarden/cli v2026.4.0 preserved from build node npm cache directories (typically ~/.npm/_cacache/ on Linux runners) — the package contents and SHA512 integrity hash confirm the specific malicious payload version and can be submitted to Checkmarx, Bitwarden, and npm security teams for coordinated disclosure and cross-organizational IOC sharing | Git diff output of .github/workflows/ directory changes across all repositories for commits made during the TeamPCP exposure window (via `git log --all --diff-filter=M --diff-filter=A --since= -- '.github/workflows/'`) — this surfaces injected workflow steps, modified secrets contexts, and any backdoor pipeline additions made using the compromised Checkmarx distribution identity's push access

Per-Action IR Details

Step 1: Containment — Immediately suspend all CI/CD pipeline jobs that reference Checkmarx KICS Docker images, the Checkmarx AST GitHub Action, Checkmarx AST Results VS Code extension, or CX Dev Assist VS Code extension. Block outbound connections to audit.checkmarx[.]cx at perimeter and endpoint DNS. Revoke all GitHub tokens, AWS SSM access keys, Azure Key Vault credentials, and GCP Secret Manager service accounts used in affected pipelines pending rotation.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST SC-7 (Boundary Protection), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 6.2 (Establish an Access Revoking Process)

Compensating: For teams without enterprise NAC/DNS filtering: push an immediate hosts file entry (0.0.0.0 audit.checkmarx.cx) via Group Policy or Ansible to all CI/CD runner nodes and developer workstations. For GitHub Actions: use the GitHub CLI (`gh secret remove`) to bulk-remove exposed secrets from affected repositories immediately. For AWS SSM keys: run `aws iam list-access-keys --user-name ` followed by `aws iam delete-access-key --access-key-id ` for each identified key. For Azure: `az ad sp credential reset --id --append false` to invalidate all existing credentials on the affected service principal. Document every revocation with a timestamp for the incident timeline.

Evidence: Before suspending pipelines, export and preserve the full GitHub Actions workflow run logs for all runs that referenced checkmarx/ast-github-action or pulled from Docker Hub under the checkmarx/kics image namespace — these logs contain the exact timestamps and injected step outputs that show whether audit.checkmarx[.]cx was contacted. Capture AWS CloudTrail 'GetParameter' and 'GetSecretValue' events (EventName filter) for the CI/CD IAM role ARNs in the exposure window; these will confirm whether SSM parameters or secrets were exfiltrated. Preserve Azure Monitor activity logs filtered on 'SecretGet' operations from the affected service principal Object ID before credential revocation destroys the attribution chain.

Step 2: Detection — Query CI/CD logs and GitHub Actions workflow history for any reference to audit.checkmarx[.]cx or @bitwarden/cli v2026.4.0. Search npm lock files (package-lock.json, yarn.lock) across all repositories for @bitwarden/cli pinned to version 2026.4.0. Review Docker pull history for KICS images pulled from Docker Hub after the campaign start date. Audit GitHub Actions workflow files for newly injected steps or unexpected uses of secrets contexts. Check cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Audit Logs) for unexpected SSM, Key Vault, or Secret Manager reads originating from CI/CD service accounts.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Run the following across all cloned repositories to locate the malicious npm package: `grep -r '"@bitwarden/cli"' . --include='package-lock.json' --include='yarn.lock' | grep '2026.4.0'`. For GitHub Actions workflow injection detection, run: `grep -r 'audit\.checkmarx' .github/workflows/ --include='*.yml' --include='*.yaml'` and separately `grep -rE 'secrets\.[A-Z_]+' .github/workflows/` to surface any unexpected secrets context references not present in baseline workflow snapshots. For Docker pull history on Linux CI runners without a SIEM, inspect `/var/lib/docker/containers/*/config.v2.json` and the Docker daemon log at `/var/log/docker.log` or `journalctl -u docker` filtering on 'checkmarx/kics'. Use `git log --all --diff-filter=A -- '.github/workflows/*.yml'` in each repository to enumerate workflow files added during the TeamPCP exposure window.

Evidence: Capture DNS query logs from all CI/CD runner hosts for resolutions of 'audit.checkmarx.cx' — on Linux runners without a DNS proxy, check systemd-resolved logs via `journalctl -u systemd-resolved | grep audit.checkmarx` or inspect network capture files if tcpdump/Wireshark was running. Pull the complete GitHub Actions artifact log bundles (downloadable as zip via GitHub API: `GET /repos/{owner}/{repo}/actions/runs/{run_id}/logs`) before GitHub's default 90-day log retention expires — these contain the raw stdout of any injected exfiltration step. Preserve the npm package tarball of @bitwarden/cli v2026.4.0 from the npm cache on affected build nodes (typically `~/.npm/_cacache/` or the runner's npm cache directory) for malware analysis and hash comparison against the legitimate Bitwarden package.

Step 3: Eradication — Replace all affected Checkmarx artifacts with versions confirmed clean by Checkmarx's official advisory (verify artifact integrity hashes before re-deployment). Remove @bitwarden/cli v2026.4.0 from all dependency manifests and replace with the verified legitimate package from the official Bitwarden registry,

confirming the source URL and hash. Audit every repository the compromised identity had push access to; revert any injected workflow files. Remove or disable VS Code extensions pending confirmed clean versions from Checkmarx.

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), NIST IR-4 (Incident Handling), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.3 (Address Unauthorized Software), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Verify Docker image integrity before re-deployment using `docker inspect --format='{{index .RepoDigests 0}}' checkmarx/kics:` and compare the SHA256 digest against the hash published in Checkmarx's official advisory. For the npm package replacement, after removing v2026.4.0 from manifests, run `npm install @bitwarden/cli@ --registry https://registry.npmjs.org` and immediately run `npm audit signatures` (available in npm v8.1+, free) to validate the package signature against the npm public key. For VS Code extension removal on developer workstations without MDM, distribute a one-liner: `code --uninstall-extension checkmarx.ast-results checkmarx.cx-dev-assist` via a PowerShell/bash script pushed through your configuration management tool or manually executed. Use `git log --author --since= -p -- '.github/workflows/'` on each repository to diff any workflow changes during the exposure window and revert injected steps using `git revert `.

Evidence: Before reverting injected workflow files, preserve the malicious workflow YAML content verbatim — capture a `git show ` of each injected commit and store the output as a forensic artifact. This preserves the injected step logic (likely curl/wget to audit.checkmarx[.]cx or an exfil command), the attacker's commit identity, and the timestamp for the incident timeline. Document the file hash (SHA256) of the @bitwarden/cli v2026.4.0 tarball cached on build nodes using `sha256sum ~/.npm/_cacache/content-v2/sha512/` — this hash is needed to confirm whether other packages in the same registry batch were similarly tampered, and may be requested by Checkmarx or law enforcement.

Step 4: Recovery — Rotate all secrets that were accessible during the exposure window: GitHub personal access tokens and Actions secrets, AWS IAM credentials with SSM access, Azure service principals with Key Vault access, GCP service accounts with Secret Manager access. Validate artifact integrity checksums against Checkmarx-published hashes before re-enabling pipelines. Re-enable pipelines in a staged manner with enhanced logging. Monitor for anomalous repository commits, unexpected workflow additions, or unusual cloud API calls for a minimum of 30 days post-remediation.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST IA-5 (Authenticator Management), NIST CP-10 (System Recovery and Reconstitution), NIST AU-12 (Audit Record Generation), CIS 5.2 (Use Unique Passwords), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 7.3 (Perform Automated Operating System Patch Management)

Compensating: For secret rotation without a secrets management platform: use the GitHub CLI to rotate Actions secrets programmatically (`gh secret set --repo --body `), and enumerate all repositories that shared the compromised secret by querying `gh api /repos/{owner}/{repo}/actions/secrets` across your org. For 30-day post-recovery monitoring without a SIEM, configure GitHub repository webhooks to POST `push`, `workflow_run`, and `create` events to a lightweight listener (e.g., a Python Flask app or smee.io proxy) and alert on any `.github/workflows/` file changes or new workflow runs referencing external Docker registries. For AWS CloudTrail without a SIEM, schedule a daily Lambda or `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetSecretValue` cron job and diff outputs against a post-rotation baseline.

Evidence: Before re-enabling any pipeline, capture a fresh baseline snapshot of all GitHub Actions workflow files (hash every `.github/workflows/*.yml` file across all repositories using `find . -path '*/.github/workflows/*.yml' -exec sha256sum {} \;`) — this signed baseline is your integrity reference for the 30-day monitoring period. For cloud provider monitoring, establish a baseline of expected `GetParameter`, `GetSecretValue`, and `SecretManagerGetSecretValue` call volumes per CI/CD service account identity from pre-incident CloudTrail/GCP Audit Logs and alert on any deviation above 2x the baseline during the monitoring window, which would indicate a persisted credential being used by TeamPCP infrastructure that survived rotation.

Step 5: Post-Incident — This attack succeeded by compromising trusted security tooling, bypassing controls that rely on source trustworthiness. Implement artifact integrity verification (e.g., Sigstore/cosign for container images, npm provenance attestation) across all CI/CD pipelines. Enforce least-privilege for CI/CD service accounts — no pipeline identity should have push access to repositories beyond its defined scope. Add DNS monitoring for typosquatted vendor domains as a standing detection control. Review the MITRE ATT&CK Supply Chain Compromise techniques (T1195.001, T1195.002) against your current control coverage and identify gaps.

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-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Risk Management), NIST AC-6 (Least Privilege), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Implement Sigstore/cosign (free, open-source) for container image verification by adding a cosign verify step to every GitHub Actions workflow that pulls a Checkmarx or other security-tooling image: `cosign verify --certificate-identity-regexp '.*checkmarx.*' --certificate-oidc-issuer https://token.actions.githubusercontent.com `. For npm provenance, add `--audit signatures` to all `npm install` steps in CI — this is a built-in npm CLI feature requiring no additional tooling. For DNS typosquatting detection as a standing control, deploy PassiveDNS logging on your perimeter (free via Zeek/Bro or Suricata with the dns.log module) and run weekly `dnstwist checkmarx.com` queries (free tool) to generate a watchlist of likely typosquats, then alert on any CI/CD runner DNS query matching that list. Map T1195.001 (Compromise Software Dependencies and Development Tools) and T1195.002 (Compromise Software Supply Chain) gaps against your pipeline inventory and document findings in the lessons-learned report required by NIST 800-61r3 §4.

Evidence: For the lessons-learned report, compile the complete attack timeline using GitHub Actions run timestamps, CloudTrail/GCP Audit Log entries, and DNS query logs to establish exactly when each Checkmarx artifact was first pulled post-compromise, how long the exposure window lasted, and which secrets were in scope during that window — this timeline is the evidentiary foundation for any regulatory breach notification assessment and for updating the IR plan per NIST IR-8 (Incident Response Plan).

Detection Guidance

Primary IOC: C2 domain audit.checkmarx[.]cx, block and alert on all DNS queries and outbound HTTP/S connections to this domain across network, endpoint, and CI/CD environments. Secondary IOC: npm package @bitwarden/cli at version 2026.4.0, scan all package-lock.json, yarn.lock, and pnpm-lock.yaml files across repositories for this exact version string. Behavioral indicators: unexpected GitHub Actions workflow modifications (new steps added to existing workflows, especially those referencing external actions or secrets); CI/CD service account calls to AWS SSM GetParameter, Azure Key Vault GetSecret, or GCP Secret Manager AccessSecretVersion outside of expected job patterns; new GitHub repository collaborators or deploy keys added following a pipeline run. Log sources to query: GitHub Actions audit log (workflow_run events, push events on .github/workflows/), AWS CloudTrail (ssm:GetParameter, secretsmanager:GetSecretValue), Azure Monitor (KeyVaultAccessPolicy, SecretGet), GCP Audit Logs (secretmanager.versions.access), Docker Hub pull history, endpoint DNS logs, and npm audit output for affected package versions.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`audit.checkmarx[.]cx`	C2 infrastructure for TeamPCP Shai-Hulud Wave 3; typosquatted domain impersonating legitimate Checkmarx vendor infrastructure; used across all attack vectors	HIGH
URL	`https://audit.checkmarx[.]cx`	C2 base URL; block all outbound connections and DNS resolution to this host	HIGH
URL	`npm:@bitwarden/cli@2026.4.0`	Malicious npm package impersonating the legitimate Bitwarden CLI; version 2026.4.0 is the confirmed malicious release identified in this campaign	HIGH

Framework Mappings

MITRE-ATTACK

- **T1059.004** — Unix Shell
- **T1555** — Credentials from Password Stores
- **T1059.007** — JavaScript
- **T1553** — Subvert Trust Controls
- **T1027** — Obfuscated Files or Information
- **T1552.004** — Private Keys
- **T1176** — Software Extensions
- **T1078.001** — Default Accounts
- **T1547** — Boot or Logon Autostart Execution
- **T1562.001** — Disable or Modify Tools
- **T1053** — Scheduled Task/Job
- **T1543** — Create or Modify System Process
- **T1078.004** — Cloud Accounts
- **T1612** — Build Image on Host
- **T1136.003** — Cloud Account
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.001** — Credentials In Files
- **T1071.001** — Web Protocols
- **T1567.001** — Exfiltration to Code Repository
- **T1567** — Exfiltration Over Web Service
- **T1195.002** — Compromise Software Supply Chain
- **T1098** — Account Manipulation

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **AC-3** — Access Enforcement
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **IA-5** — Authenticator Management
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

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

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.004	Unix Shell	Execution

Technique ID	Technique Name	Tactic
T1555	Credentials from Password Stores	Credential-Access
T1059.007	JavaScript	Execution
T1553	Subvert Trust Controls	Defense-Evasion
T1027	Obfuscated Files or Information	Defense-Evasion
T1552.004	Private Keys	Credential-Access
T1176	Software Extensions	Persistence
T1078.001	Default Accounts	Defense-Evasion
T1547	Boot or Logon Autostart Execution	Persistence
T1562.001	Disable or Modify Tools	Defense-Evasion
T1053	Scheduled Task/Job	Execution
T1543	Create or Modify System Process	Persistence
T1078.004	Cloud Accounts	Defense-Evasion
T1612	Build Image on Host	Defense-Evasion
T1136.003	Cloud Account	Persistence
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1071.001	Web Protocols	Command-And-Control
T1567.001	Exfiltration to Code Repository	Exfiltration
T1567	Exfiltration Over Web Service	Exfiltration
T1195.002	Compromise Software Supply Chain	Initial-Access
T1098	Account Manipulation	Persistence

Sources

Source	URL	Tier
Unit 42	https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-att...	T3
	https://www.microsoft.com/en-us/security/blog/2025/12/09/shai-hulud...	T1

Source	URL	Tier
	https://unit42.paloaltonetworks.com/npm-supply-chain-attack/	T3
	https://www.morphisec.com/blog/supply-chain-attack-mitigation/	T3
Malicious KICS Docker Images and VS Code Extensions Hit ...	https://thehackernews.com/2026/04/malicious-kics-docker-images-and-...	T3

Generated 2026-04-25 13:52 UTC by TJS Security Command Center