INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-02 13:41 UTC

TeamPCP's Shai-Hulud Campaign Reaches Enterprise Scale: SAP, Bitwarden, and Checkmarx Toolchains Compromised in Coordinated Worm Wave

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0260
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm ecosystem: @cap-js/sqlite (v2.2.2), @cap-js/postgres, @cap-js/db-service (v2.10.1), mbt (SAP CAP Model); @bitwarden/cli; Checkmarx KICS Docker images, ast-github-action, ast-results, cx-dev-assist VS Code extensions; GitHub Actions; AWS, Azure, GCP, Kubernetes environments; Axios
Published	2026-05-02T00:10:33+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

In April 2026, the threat actor TeamPCP injected credential-stealing malware into npm packages with approximately 570,000 combined weekly downloads, compromising SAP Cloud Application Programming Model libraries, the Bitwarden CLI, and Checkmarx security tooling. The malware self-propagates by republishing infected package versions to npm, extending its reach across dependent CI/CD pipelines and cloud environments. Organizations using these toolchains face direct risk of cloud credential theft (AWS, Azure, GCP), Kubernetes secret exposure, and GitHub token compromise, enabling attackers to pivot into production infrastructure.

Technical Analysis

TeamPCP's Shai-Hulud campaign executed two successive npm supply chain attack waves in April 2026, building on a pattern first documented in September 2025. Confirmed compromised packages and versions: @cap-js/sqlite v2.2.2, @cap-js/db-service v2.10.1, and the mbt (SAP CAP Model) package; @bitwarden/cli; and Checkmarx assets including KICS Docker images, ast-github-action, ast-results GitHub Action, and the cx-dev-assist VS Code extension. The malicious payload is a self-propagating credential stealer. Once executed in a build environment, it targets CI/CD pipeline secrets, cloud provider credentials (AWS, Azure, GCP),

Kubernetes secrets, and GitHub tokens, then republishes infected package versions to npm to propagate downstream. Deliberate targeting of Checkmarx tooling, a widely deployed security scanning platform, suggests the actor specifically sought to compromise security pipeline infrastructure to both maximize credential harvest scope and suppress detection. Reused and refined tooling across both waves indicates an organized, persistent operation. A concurrent April 2026 npm supply chain campaign also targeted Axios (Microsoft Security Blog); investigate whether the two campaigns are coordinated or independent. Relevant CWEs: CWE-522 (Insufficiently Protected Credentials), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-798 (Use of Hard-coded Credentials), CWE-494 (Download of Code Without Integrity Check), CWE-506 (Embedded Malicious Code). MITRE ATT&CK coverage includes T1195.001 (Supply Chain Compromise: Compromise Software Dependencies), T1552.001 and T1552.004 (Credential Access from files and CI/CD), T1567.001 (Exfiltration to Code Repository), T1059.007 (JavaScript execution), and T1027 (Obfuscated Files). No CVE has been assigned; the CVSS 9.5 base score is an editorial estimate based on scope and impact, not an official NVD or vendor rating.

Action Checklist

1. Step 1: Containment. Immediately audit all build environments, CI/CD pipelines, and developer workstations for the presence of @cap-js/sqlite v2.2.2, @cap-js/db-service v2.10.1, mbt (SAP CAP Model), @bitwarden/cli, Checkmarx KICS Docker images (verify digests against the official Checkmarx GitHub repository: https://github.com/Checkmarx/kics), ast-github-action, ast-results, and cx-dev-assist. Isolate any pipeline that executed these packages from production systems and cloud credential stores pending investigation. Block installation of the identified malicious versions via npm audit enforcement or private registry allow-listing.
2. Step 2: Detection. Review CI/CD pipeline logs for outbound network connections to unexpected external endpoints during npm install or build phases. Query npm audit logs and lock file histories for presence of the compromised package versions. Check GitHub Actions workflow run logs for ast-github-action and ast-results executions. Inspect VS Code extension logs for cx-dev-assist activity. Look for anomalous API calls using cloud credentials (AWS CloudTrail, Azure Activity Log, GCP Cloud Audit Logs) originating from build systems. Search for evidence of npm publish events from build accounts that were not authorized. Behavioral indicator: npm packages republishing themselves or unexpected new versions appearing in your private registry mirror.
3. Step 3: Eradication. Pin all affected packages to known-clean versions and verify package integrity using published checksums from the SAP community advisory and npm registry. Remove and rotate all secrets, tokens, and credentials that were accessible in any environment where compromised packages executed: AWS IAM keys, Azure service principal secrets, GCP service account keys, Kubernetes secrets, and GitHub personal access tokens or Actions secrets. Revoke and reissue affected Bitwarden CLI API keys. Rebuild CI/CD runner images from verified base images. Remove the cx-dev-assist VS Code extension and replace with a version verified against the Checkmarx official release hash (https://github.com/Checkmarx/cx-dev-assist).
4. Step 4: Recovery. After credential rotation, validate that no unauthorized access occurred by reviewing cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Audit Logs) for the period the compromised packages were in use. Confirm that no new IAM roles, users, or service accounts were created by attackers. Re-run Checkmarx and third-party SAST scans using verified-clean tooling before resuming production deployments. Monitor npm publish activity from your organization's accounts for any unauthorized publications. Restore CI/CD pipelines only after confirming clean build environments.

5. Step 5: Post-Incident. This campaign exploited the absence of package integrity verification and open access to third-party packages in CI/CD pipelines. Implement Software Composition Analysis (SCA) with integrity verification (SLSA framework, npm package-lock enforcement, or private registry mirroring with hash pinning) as a pipeline gate. Adopt least-privilege secret scoping in CI/CD so pipeline secrets are scoped to specific jobs and auto-expire. Add monitoring for anomalous npm publish events from build identities. Review whether security tooling (SAST, DAST scanners, VS Code extensions) is subject to the same software supply chain controls as application dependencies; this campaign demonstrates that security tooling is an active target.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to executive leadership, legal counsel, and initiate breach notification assessment immediately if CloudTrail, Azure Activity Log, or GCP Audit Log evidence confirms that stolen credentials were used to access, exfiltrate, or modify data in production environments, or if unauthorized IAM resources created during the exposure window cannot be fully accounted for — either condition may trigger regulatory breach notification obligations depending on data classification and jurisdiction.
Recovery Notes	Recovery cannot be declared complete until all cloud credentials accessible during the exposure window have been rotated and the post-rotation IAM state has been diffed against a pre-incident baseline to rule out attacker-created persistence (roles, users, service accounts, or OIDC trust relationships). Monitor npm publish activity for your organization's scoped packages and your private registry mirror for a minimum of 30 days post-eradication, as the Shai-Hulud campaign's self-propagation mechanism may have seeded dependent packages that have not yet been identified. Do not restore Checkmarx KICS, ast-github-action, ast-results, or cx-dev-assist to any pipeline until each artifact has been verified by SHA256 digest against Checkmarx's official published release hashes — security tooling that cannot be integrity-verified must be treated as untrusted.

Forensic Artifacts	npm postinstall execution traces in ~/.npm/_logs/ on CI runners and developer workstations: the TeamPCP malware in @cap-js/sqlite v2.2.2 and @cap-js/db-service v2.10.1 executes its credential-stealing payload via a postinstall lifecycle hook, producing a timestamped entry in these logs that records the script invocation and any stdout/stderr output including attempted exfiltration endpoints | AWS CloudTrail events for sts:GetCallerIdentity, iam:ListAccessKeys, iam:CreateAccessKey, and s3:ListBuckets sourced from CI runner IP addresses during the compromise window: these are the specific API call sequence consistent with automated cloud credential harvesting malware targeting AWS environments via stolen IAM key material from build environments | Kubernetes secret access audit events in the API server audit log (audit.log) for verbs 'get' and 'list' on resource 'secrets' in all namespaces, filtered to the service account or kubeconfig identity used by CI runners: the worm's credential harvesting targets Kubernetes secrets stores accessible from build environments, and this log records whether that access occurred | npm registry publish event logs (from Verdaccio, Artifactory, or Nexus) or GitHub Packages audit events showing any package version published under your organization's npm scope during the exposure window from a build system identity: the Shai-Hulud self-propagation mechanism republishes infected versions to extend reach, and unauthorized publications from CI service accounts are the definitive indicator of active propagation | Bitwarden CLI session cache file at ~/.config/Bitwarden CLI/data.json (Linux/macOS) or %APPDATA%\Bitwarden CLI\data.json (Windows) timestamped during the malicious @bitwarden/cli execution window: this file contains the encrypted vault session token and, if the vault was unlocked during the compromise period, represents the credential surface accessible to the malware's exfiltration routine

Per-Action IR Details

Step 1: Containment — Immediately audit all build environments, CI/CD pipelines, and developer workstations for the presence of @cap-js/sqlite v2.2.2, @cap-js/db-service v2.10.1, mbt (SAP CAP Model), @bitwarden/cli, Checkmarx KICS Docker images (unverified digests), ast-github-action, ast-results, and cx-dev-assist. Isolate any pipeline that executed these packages from production systems and cloud credential stores pending investigation. Block installation of the identified malicious versions via npm audit enforcement or private registry allow-listing.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-2 (Baseline Configuration), NIST SI-3 (Malicious Code Protection), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run `npm ls --all 2>/dev/null | grep -E '@cap-js/sqlite|@cap-js/db-service|@bitwarden/cli|mbt'` in each build workspace root to enumerate installed versions. For CI/CD runners, extract the npm lockfile: `cat package-lock.json | python3 -m json.tool | grep -A2 -E 'cap-js/sqlite|cap-js/db-service|bitwarden/cli|mbt'`. For Checkmarx KICS Docker images, run `docker images --digests | grep kics` and compare digests against Checkmarx's published known-good SHA256 values. Revoke network egress from affected runner hosts immediately using `iptables -I OUTPUT -m owner --uid-owner -j DROP` as an emergency measure while formal isolation is arranged.

Evidence: Before isolating any pipeline, snapshot the following: (1) Full `package-lock.json` and `yarn.lock` files from every affected workspace — these record the exact resolved versions of @cap-js/sqlite, @cap-js/db-service, and @bitwarden/cli that were installed, including sub-dependency chains that may carry the malicious payload. (2) Docker image layer manifests for any Checkmarx KICS image in use: `docker inspect --format='{{json .RootFS}}'` captures the layer SHA256 chain needed to confirm whether an unverified digest is present. (3) GitHub Actions workflow run artifacts and runner environment exports (`printenv > runner_env_snapshot.txt`) taken before credential rotation — these preserve evidence of which secrets were in scope during a malicious build execution.

Step 2: Detection — Review CI/CD pipeline logs for outbound network connections to unexpected external endpoints during npm install or build phases. Query npm audit logs and lock file histories for presence of the compromised package versions. Check GitHub Actions workflow run logs for ast-github-action and ast-results executions. Inspect VS Code extension logs for cx-dev-assist activity. Look for anomalous API calls using cloud credentials (AWS CloudTrail, Azure Activity Log, GCP Cloud Audit Logs) originating from build systems. Search for evidence of npm publish events from build accounts that were not authorized. Behavioral indicator: npm packages republishing themselves or unexpected new versions appearing in your private registry mirror.

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-12 (Audit Record Generation), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For egress detection without a SIEM: on Linux CI runners, use `ss -tnp` or `netstat -tnp` captured during a build execution, or replay historical netflow if available. For AWS, run: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventSource,AttributeValue=sts.amazonaws.com --start-time --end-time ` to surface AssumeRole or GetCallerIdentity calls from build host IPs. For GitHub Actions, use the GitHub API: `gh api /repos/{owner}/{repo}/actions/runs --jq '.workflow_runs[] | select(.name | test("ast-github-action|ast-results")) | {id, conclusion, created_at, head_sha}'` to enumerate all executions of the compromised actions during the exposure window. For VS Code cx-dev-assist, check `~/.vscode/extensions/checkmarx.cx-dev-assist-*/` for extension files modified after the known malicious release date and review `~/.vscode/logs/` for outbound request traces.

Evidence: Capture before analysis concludes: (1) AWS CloudTrail events for event names `sts:GetCallerIdentity`, `iam:ListUsers`, `iam:CreateAccessKey`, `s3:ListBuckets` sourced from your CI runner IP ranges during the TeamPCP exposure window — these are the reconnaissance and exfiltration API calls consistent with credential-stealing malware targeting AWS. (2) GCP Cloud Audit Logs filtered on `protoPayload.authenticationInfo.principalEmail` matching CI service accounts, specifically for `storage.objects.list`, `iam.serviceAccounts.keys.create`, and `container.clusters.list` — GCP artifact consistent with cloud-targeting worm behavior. (3) npm registry event logs from your private registry mirror (Verdaccio, Artifactory, or Nexus) showing any `npm publish` events originating from build machine service accounts — the self-propagation mechanism of the Shai-Hulud campaign would surface here as unauthorized version publications. (4) GitHub audit log events of type `packages.package_version_published` or `workflow_job` filtered to the compromised action identifiers.

Step 3: Eradication — Pin all affected packages to known-clean versions and verify package integrity using published checksums from the SAP community advisory and npm registry. Remove and rotate all secrets, tokens, and credentials that were accessible in any environment where compromised packages executed: AWS IAM keys, Azure service principal secrets, GCP service account keys, Kubernetes secrets, and GitHub personal access tokens or Actions secrets. Revoke and reissue affected Bitwarden CLI API keys. Rebuild CI/CD runner images from verified base images. Remove the cx-dev-assist VS Code extension and replace with a version verified against the Checkmarx official release hash.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST IA-5 (Authenticator Management), CIS 5.2 (Use Unique Passwords), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Verify npm package integrity without enterprise tooling: `npm view @cap-js/sqlite@ dist.integrity` returns the expected sha512 hash; compare against your locally cached package using `cat node_modules/@cap-js/sqlite/package.json | grep _integrity`. For Kubernetes secrets rotation: `kubectl get secrets --all-namespaces -o json | jq '.items[] | select(.metadata.annotations["last-rotated"] == null or (.metadata.annotations["last-rotated"] | fromdateiso8601) )' ` to identify unrotated secrets. For GitHub Actions secrets, use `gh secret list --repo {owner}/{repo}` to enumerate all secrets that were in scope during affected workflow runs,

then rotate via `gh secret set --body ""`. Rebuild Docker-based CI runners from a pinned base image hash: `docker build --no-cache --pull -t ci-runner:clean .` to ensure no layer cache from a compromised image persists.

Evidence: Before overwriting or rotating, preserve: (1) A memory snapshot of any long-running CI/CD runner process that executed the malicious packages — use `gcore ` on Linux to capture a core dump, which may contain cloud credentials or exfiltration endpoints in memory that are not yet in logs. (2) The full contents of `~/.npm/_logs/` on developer workstations and runner hosts, which record the npm install execution trace including any postinstall script output — TeamPCP's malware likely executes via a `postinstall` hook in the compromised packages, and this log captures its invocation. (3) Bitwarden CLI session tokens cached at `~/.config/Bitwarden CLI/data.json` (Linux/macOS) or `%APPDATA%\Bitwarden CLI\data.json` (Windows) before invalidation — these confirm whether an active unlocked session was present during malicious package execution.

Step 4: Recovery — After credential rotation, validate that no unauthorized access occurred by reviewing cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Audit Logs) for the period the compromised packages were in use. Confirm that no new IAM roles, users, or service accounts were created by attackers. Re-run Checkmarx and third-party SAST scans using verified-clean tooling before resuming production deployments. Monitor npm publish activity from your organization's accounts for any unauthorized publications. Restore CI/CD pipelines only after confirming clean build environments.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-11 (Audit Record Retention), NIST SI-6 (Security and Privacy Function Verification), NIST CM-2 (Baseline Configuration), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 5.1 (Establish and Maintain an Inventory of Accounts)

Compensating: Enumerate potentially attacker-created IAM resources without a CSPM tool: AWS: `aws iam list-users --query 'Users[?CreateDate>=``]'` and `aws iam list-roles --query 'Roles[?CreateDate>=``]'`. GCP: `gcloud iam service-accounts list --filter='createTime>='`. Azure: `az ad sp list --query '[?additionalProperties.createdDateTime>=``]' --all`. For npm publish monitoring, configure a webhook on your Verdaccio or Artifactory instance to POST on any publish event, then pipe to a simple alerting script: `curl -s https://registry.npmjs.org/-/npm/v1/security/audits/quick -d @package-lock.json -H 'Content-Type: application/json' | jq '.advisories'` for ongoing advisory checks against the cleaned lockfile.

Evidence: Before declaring recovery complete, retain: (1) A full export of AWS CloudTrail, Azure Activity Log, and GCP Audit Log events scoped to the entire compromise window (from earliest known @cap-js/sqlite v2.2.2 or @bitwarden/cli installation date through credential rotation completion) — these constitute the primary forensic record of attacker activity and are required for breach notification analysis. (2) Snapshots of current IAM state post-rotation (user list, role list, service account list, attached policies) compared via diff against pre-incident baselines — delta entries are candidate attacker-created persistence mechanisms. (3) npm registry publish logs for your organization's scoped packages covering the compromise window, confirming whether the Shai-Hulud self-propagation mechanism successfully republished any packages under your organization's namespace.

Step 5: Post-Incident — This campaign exploited the absence of package integrity verification and open access to third-party packages in CI/CD pipelines. Implement Software Composition Analysis (SCA) with integrity verification (SLSA framework, npm package-lock enforcement, or private registry mirroring with hash pinning) as a pipeline gate. Adopt least-privilege secret scoping in CI/CD so pipeline secrets are scoped to specific jobs and auto-expire. Add monitoring for anomalous npm publish events from build identities. Review whether security tooling (SAST, DAST scanners, VS Code extensions) is subject to the same software supply chain controls as application dependencies — this campaign demonstrates that security tooling is an active target.

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 SA-12 (Supply Chain Protection), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-2 (Baseline Configuration), 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), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Implement SLSA-aligned integrity verification without enterprise tooling: enforce `npm ci` (not `npm install`) in all pipelines — this requires a committed lockfile and refuses installation if checksums do not match. Add a pre-build pipeline step using `npm audit --audit-level=high` against a private registry mirror (Verdaccio is free and self-hostable) configured to allow-list only pinned package versions by hash. For secret scoping, replace long-lived AWS IAM keys in GitHub Actions with OIDC federation: `aws sts assume-role-with-web-identity` grants ephemeral credentials scoped to a single workflow job with a configurable max session duration, eliminating the long-lived key theft vector that TeamPCP exploited. For anomalous publish detection, configure a free npm webhook via `npm hook add ` to alert on any publish event under your namespace.

Evidence: For the post-incident lessons-learned record, preserve: (1) The complete timeline of @cap-js/sqlite v2.2.2, @cap-js/db-service v2.10.1, and @bitwarden/cli malicious version availability on npm (first seen, peak download volume, takedown timestamp) correlated against your internal first-install dates — this establishes the definitive exposure window for breach notification and regulatory reporting purposes. (2) A dependency tree export (`npm ls --all --json > full_dependency_tree_.json`) from each affected project, documenting the transitive dependency paths through which the malicious packages reached your builds — this is required to assess whether any internal packages inadvertently became carriers in the Shai-Hulud self-propagation chain. (3) The VS Code extension manifest and installed file hashes for cx-dev-assist captured at discovery time — developer workstation tooling is typically excluded from supply chain audits, and this evidence supports the process control gap finding.

Detection Guidance

Primary detection surface is CI/CD pipeline and build environment logs. Query for: (1) npm install events referencing @cap-js/sqlite@2.2.2, @cap-js/db-service@2.10.1, or @bitwarden/cli in any version installed after April 1, 2026 pending clean version confirmation. (2) Outbound HTTP/S connections from build runners to domains or IPs not in your approved egress list, particularly during the npm install or test phases; exfiltration uses T1071.001 (HTTP) and T1041. (3) AWS CloudTrail: ListSecrets, GetSecretValue, DescribeInstances, or AssumeRole events from EC2 or Lambda identities associated with build infrastructure. (4) Azure Activity Log: key vault secret read events from service principals assigned to build pipelines. (5) GCP Cloud Audit Logs: secretmanager.versions.access from build service accounts. (6) Kubernetes: kubectl get secrets or API server audit log events for secret reads from non-standard service accounts. (7) GitHub: review Actions secrets access logs and any unexpected npm publish events from Actions workflows. (8) npm registry: check your organization's publish history for unauthorized package versions. Behavioral IOC: npm packages self-republishing, any unexpected new patch or minor version appearing from a dependency without a corresponding upstream release. Checkmarx-specific: verify Docker image digests for KICS against the official Checkmarx container registry (ghcr.io/checkmarx/kics) and GitHub release page (https://github.com/Checkmarx/kics/releases); compare ast-github-action and ast-results Action SHAs against the official Checkmarx GitHub repository commit history (https://github.com/Checkmarx/ast-github-action and https://github.com/Checkmarx/ast-results).

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`not confirmed in available sources`	Malicious package versions: @cap-js/sqlite@2.2.2, @cap-js/db-service@2.10.1 — verify package integrity hashes against SAP community advisory and npm registry	LOW
URL	`https://community.sap.com/t5/technology-q-a/compromised-npm-packages-cap-js-sqlite-2-2-2-cap-js-db-service-2-10-1-cap/qaq-p/14387231`	SAP community advisory listing confirmed compromised package versions — check for updated IOC list	HIGH
URL	`https://unit42.paloaltonetworks.com/npm-supply-chain-attack/`	Unit 42 technical reporting on the Shai-Hulud campaign — may contain additional IOCs; verify URL resolves at access time	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1650** — Acquire Access
- **T1552.004** — Private Keys
- **T1057** — Process Discovery
- **T1543** — Create or Modify System Process
- **T1041** — Exfiltration Over C2 Channel
- **T1071.001** — Web Protocols
- **T1059.007** — JavaScript
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1078.001** — Default Accounts
- **T1552.001** — Credentials In Files
- **T1567.001** — Exfiltration to Code Repository
- **T1053** — Scheduled Task/Job
- **T1078** — Valid Accounts
- **T1059** — Command and Scripting Interpreter
- **T1027** — Obfuscated Files or Information

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **AC-3** — Access Enforcement
- **CM-7** — Least Functionality

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **SI-3** — Malicious Code Protection
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **5.2** — Use Unique Passwords
- **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
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication

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

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
- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1650	Acquire Access	Resource-Development
T1552.004	Private Keys	Credential-Access
T1057	Process Discovery	Discovery
T1543	Create or Modify System Process	Persistence
T1041	Exfiltration Over C2 Channel	Exfiltration
T1071.001	Web Protocols	Command-And-Control
T1059.007	JavaScript	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1078.001	Default Accounts	Defense-Evasion
T1552.001	Credentials In Files	Credential-Access
T1567.001	Exfiltration to Code Repository	Exfiltration
T1053	Scheduled Task/Job	Execution
T1078	Valid Accounts	Defense-Evasion
T1059	Command and Scripting Interpreter	Execution
T1027	Obfuscated Files or Information	Defense-Evasion

Sources

Source	URL	Tier
Unit 42	https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-att...	T3
	https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-att...	T3
	https://www.microsoft.com/en-us/security/blog/2026/04/01/mitigating...	T1
	https://unit42.paloaltonetworks.com/npm-supply-chain-attack/	T3
Compromised npm packages: @cap-js/sqlite@2.2.2, @c...	https://community.sap.com/t5/technology-q-a/compromised-npm-package.. .	T3

DISCLAIMER

Generated 2026-05-02 13:41 UTC by TJS Security Command Center