INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-24 06:45 UTC

TeamPCP Weaponizes Bitwarden CLI npm Package to Harvest CI/CD Secrets and Self-Propagate Across Developer Ecosystems

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0213
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	@bitwarden/cli npm package v2026.4.0; Checkmarx KICS dev tooling; GitHub Actions CI/CD pipelines; cloud credential stores (AWS, Azure, GCP); npm ecosystem downstream packages
Published	2026-04-23T15:21:01
Discovery Source	Rss

Executive Summary

On April 22, 2026, threat actor TeamPCP published a malicious version of the @bitwarden/cli npm package (v2026.4.0) that remained publicly available for approximately 90 minutes before removal. Any developer or CI/CD pipeline that installed this version was exposed to exfiltration of npm authentication tokens, SSH keys, and cloud credentials for AWS, Azure, and Google Cloud Platform. A self-propagation mechanism means organizations that installed the package may have unknowingly poisoned downstream packages they control, extending blast radius well beyond direct victims.

Technical Analysis

Affected package: @bitwarden/cli v2026.4.0 (npm). Attack vector: TeamPCP compromised a Checkmarx development tool and abused Bitwarden's own CI/CD pipeline to publish the trojanized package. The malicious version embedded JavaScript-based malware (CWE-506) that executed via npm preinstall scripts (MITRE T1543) to harvest credentials stored in files and environment variables (T1552.001, CWE-312, CWE-522), targeting npm tokens, SSH keys, and cloud provider credentials (AWS, Azure, GCP). Exfiltration used web service channels (T1567.001). A self-propagation routine (T1195.001) seeded malicious code into downstream npm packages owned by any developer who ran the compromised version, compounding exposure. Obfuscation techniques (T1027) were employed to hinder static analysis. Infrastructure and obfuscation patterns overlap with an ongoing Checkmarx supply chain breach, indicating a coordinated multi-target operation. No CVE assigned. No official CVSS vector confirmed; qualitative severity assessed as critical based

on attack scope and blast radius. Relevant CWEs: CWE-506, CWE-494, CWE-312, CWE-522. MITRE techniques: T1195.001, T1543, T1552.001, T1552.004, T1567.001, T1027, T1059.007, T1554, T1650, T1078.004. Source confidence: medium. Primary technical analysis from JFrog Security and Ox Security vendor research sources; formal confirmation from Bitwarden/NVD pending.

Action Checklist

1. Step 1: Containment, immediately identify any system, pipeline, or developer environment that installed @bitwarden/cli v2026.4.0. Isolate affected build agents and developer workstations from production networks and credential stores. Revoke all npm authentication tokens, SSH keys, and cloud IAM credentials (AWS access keys, Azure service principals, GCP service accounts) that were present on any system where v2026.4.0 executed. Treat all credentials accessible to those environments as fully compromised.
2. Step 2: Detection, audit npm install logs, CI/CD pipeline logs (GitHub Actions run history, build system artifact logs), and package-lock.json or yarn.lock files across all repositories for @bitwarden/cli@2026.4.0. Review npm audit logs for outbound POST requests to unexpected domains during or after package installation. Check cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Cloud Audit Logs) for anomalous API calls following the April 22, 2026 window. Inspect downstream npm packages your organization publishes for unexpected modifications to package contents or preinstall scripts introduced after April 22.
3. Step 3: Eradication, upgrade @bitwarden/cli to the latest confirmed-clean release per Bitwarden's official security advisory and GitHub releases page. Remove v2026.4.0 from all package caches, artifact registries (JFrog Artifactory, Nexus, GitHub Packages), and lock files. Audit all npm packages your organization publishes for injected malicious code introduced by the self-propagation mechanism; republish clean versions with updated checksums. Reissue all rotated credentials with least-privilege scopes.
4. Step 4: Recovery, validate that CI/CD pipelines are pulling verified clean package versions by confirming SHA integrity hashes against Bitwarden's official release. Re-enable production deployments only after credential rotation is confirmed complete and downstream packages have been audited. Monitor cloud provider access logs and npm publish activity for at least 30 days post-remediation for signs of persistent access using credentials that may have been exfiltrated before detection.
5. Step 5: Post-Incident, this attack exploited the absence of cryptographic integrity verification on npm package installs (CWE-494) and broad credential availability in CI/CD environments. Control improvements to implement: enforce npm package integrity verification with lockfile pinning and Subresource Integrity checks; adopt short-lived, scoped CI/CD credentials using OIDC-based authentication (e.g., GitHub Actions OIDC for AWS/Azure/GCP) instead of long-lived static keys; implement a private package mirror or registry proxy with allow-listing and integrity scanning; add software composition analysis (SCA) tooling to CI/CD pipelines to flag unexpected package version changes before installation.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO, legal counsel, and external IR retainer immediately if AWS CloudTrail, Azure Monitor, or GCP Cloud Audit Logs confirm any API calls using rotated credentials after the April 22 exposure window, if any downstream package your organization publishes shows evidence of self-propagation injection (triggering potential customer breach notification obligations), or if the 2-person team lacks capacity to complete credential rotation across all three cloud providers within 4 hours of confirmed exposure.
Recovery Notes	Re-enable production CI/CD pipelines only after a verified `npm ci` run with lockfile integrity checks confirms the clean @bitwarden/cli version resolves with a SHA matching Bitwarden's published release, and after all cloud IAM credentials present on affected systems have been rotated and the old credentials confirmed disabled or deleted — not merely deactivated. Maintain enhanced monitoring of AWS CloudTrail, Azure Monitor, and GCP Cloud Audit Logs, plus npm publish webhook alerts for your organization's packages, for a minimum of 30 days given that TeamPCP may have exfiltrated credentials during the 90-minute window before removal and could attempt delayed use to avoid correlation with the April 22 incident. Any anomalous API call from a newly rotated credential or unexpected npm publish event for an org-owned package during the 30-day window should be treated as an active incident, not a false positive.
Forensic Artifacts	npm debug logs at `~/.npm/_logs/*.log` on affected developer workstations and CI/CD runner home directories — these capture the full install lifecycle for @bitwarden/cli v2026.4.0 including postinstall script execution, subprocess spawning, and any DNS/HTTP activity initiated by the malicious payload during the TeamPCP exfiltration phase | GitHub Actions workflow run logs (downloadable via `gh run download`) for all runs in the 2026-04-22T00:00Z to 2026-04-22T02:30Z window — specifically the stdout of npm install steps and any subsequent runner-level network activity, which will show whether the v2026.4.0 postinstall hook executed and whether it successfully reached TeamPCP exfiltration infrastructure | AWS CloudTrail event history filtered for `iam:CreateAccessKey`, `secretsmanager:GetSecretValue`, `sts:AssumeRole`, and `s3:ListBuckets` events originating from CI/CD runner IP ranges or from unfamiliar source IPs after April 22 — these would evidence successful credential harvest and subsequent unauthorized use by TeamPCP | Filesystem diff of `node_modules/@bitwarden/cli/` directory captured immediately at containment versus the known-clean package contents from Bitwarden's GitHub release tag — the delta isolates the malicious code injected by TeamPCP, including the specific JS files modified to implement the credential harvesting and self-propagation logic targeting `process.env` and local SSH agent sockets | Git commit history (`git log --all -p -- package.json package-lock.json`) for all npm packages your organization publishes, covering the period from April 22 onward — commits not attributable to known maintainer identities that modify `scripts.preinstall`, `scripts.postinstall`, or add new dependencies are direct evidence of the TeamPCP self-propagation mechanism having successfully injected into your published packages

Per-Action IR Details

Step 1: Containment — immediately identify any system, pipeline, or developer environment that installed @bitwarden/cli v2026.4.0. Isolate affected build agents and developer workstations from production networks and credential stores. Revoke all npm authentication tokens, SSH keys, and cloud IAM credentials (AWS access keys, Azure service principals, GCP service accounts) that were present on any system where v2026.4.0 executed. Treat all credentials accessible to those environments as fully compromised.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST IA-5 (Authenticator Management), CIS 5.3 (Disable Dormant Accounts), CIS 6.2 (Establish an Access Revoking Process), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run `npm ls @bitwarden/cli --all --depth=0` on each developer workstation and build agent to confirm installed version. On GitHub Actions, pull run logs via `gh run list --json databaseId,conclusion,createdAt | gh run view --log` filtering for any run between 2026-04-22T00:00Z and 2026-04-22T02:30Z that references @bitwarden/cli. For AWS, immediately call `aws iam list-access-keys --user-name ` and `aws iam delete-access-key` for any key present on affected hosts; for GCP, use `gcloud iam service-accounts keys list` and `gcloud iam service-accounts keys delete`. Revoke npm tokens via `npm token revoke ` for each token discoverable in `~/.npmrc` or CI/CD secret stores.

Evidence: BEFORE isolating systems, capture: (1) full contents of `~/.npmrc` and any project-level `.npmrc` files on affected workstations — these contain the npm auth tokens that v2026.4.0's exfiltration payload targeted; (2) `node_modules/@bitwarden/cli/` directory tree and file hashes (`sha256sum node_modules/@bitwarden/cli/package.json` and all JS entry points) to preserve the malicious package artifact before remediation overwrites it; (3) process tree snapshot (`ps auxf` on Linux, `Get-Process` with parent PID on Windows) taken immediately to capture any residual node process spawned by the package's postinstall script; (4) outbound network connection state (`ss -tnp` or `netstat -anob`) to identify any live exfiltration channel to TeamPCP infrastructure still open at time of containment.

Step 2: Detection — audit npm install logs, CI/CD pipeline logs (GitHub Actions run history, build system artifact logs), and package-lock.json or yarn.lock files across all repositories for @bitwarden/cli@2026.4.0. Review npm audit logs for outbound POST requests to unexpected domains during or after package installation. Check cloud provider access logs (AWS CloudTrail, Azure Monitor, GCP Cloud Audit Logs) for anomalous API calls following the April 22, 2026 window. Inspect downstream npm packages your organization publishes for unexpected modifications to package contents or preinstall scripts introduced after April 22.

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: Scan all repositories for the poisoned lockfile entry using: `grep -r '"@bitwarden/cli"' --include='package-lock.json' --include='yarn.lock' /path/to/repos | grep '2026.4.0'`. For GitHub Actions, use the GitHub CLI: `gh api /repos/{owner}/{repo}/actions/runs --jq '.workflow_runs[] | select(.created_at >= "2026-04-22T00:00:00Z" and .created_at <= "2026-04-22T02:30:00Z")' ` to enumerate runs in the exposure window. For AWS CloudTrail, run: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventSource,AttributeValue=iam.amazonaws.com --start-time 2026-04-22T00:00:00Z --end-time 2026-04-23T00:00:00Z` filtering for `CreateAccessKey`, `GetSecretValue`, or `AssumeRole` events from unusual source IPs. For outbound exfiltration detection without a SIEM, deploy the Sigma rule for suspicious npm postinstall network activity against syslog or auditd logs using `sigma convert -t grep` and apply to `/var/log/audit/audit.log` filtering for `execve` syscalls from node processes followed by outbound connections on port 443 to non-npm-registry destinations.

Evidence: Capture BEFORE analysis consumes or overwrites logs: (1) GitHub Actions raw log archives for all workflow runs between 2026-04-22T00:00Z and 2026-04-22T02:30Z — download via `gh run download ` — specifically looking for `npm install` steps that resolved @bitwarden/cli and any subsequent `curl`, `wget`, or `node` subprocess activity visible in runner stdout; (2) AWS CloudTrail S3 bucket exports for the April 22 window, specifically `iam:CreateAccessKey`, `secretsmanager:GetSecretValue`, `sts:AssumeRole`, and `s3:GetObject` events originating from CI/CD runner IP ranges; (3) npm debug log at `~/.npm/_logs/` on each affected developer machine, which captures resolved package URLs and install lifecycle script execution including the malicious postinstall hook from v2026.4.0; (4) `git log --all --diff-filter=M -- package.json package-lock.json` output for each downstream package repository your organization maintains, to identify unauthorized commits to these files after April 22 that may reflect the self-propagation mechanism injecting itself into packages you publish.

Step 3: Eradication — upgrade @bitwarden/cli to the latest confirmed-clean release per Bitwarden's official advisory (verify the clean version at https://github.com/bitwarden/clients before installing). Remove v2026.4.0 from all package caches, artifact registries (JFrog Artifactory, Nexus, GitHub Packages), and lock files. Audit all npm packages your organization publishes for injected malicious code introduced by the self-propagation mechanism; republish clean versions with updated checksums. Reissue all rotated credentials with least-privilege scopes.

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 IA-5 (Authenticator Management), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Clear the local npm cache on all affected systems with `npm cache clean --force` and verify with `npm cache verify`. For JFrog Artifactory, use the REST API: `curl -u admin:password -X DELETE 'https:///artifactory/npm-local/@bitwarden/cli/-/@bitwarden/cli-2026.4.0.tgz'`. For GitHub Packages, delete via `gh api --method DELETE /orgs/{org}/packages/npm/%40bitwarden%2Fcli/versions/{version_id}`. To audit your organization's published npm packages for the self-propagation payload, diff the current package contents against the last known-good git tag: `npm pack && tar -xzf *.tgz && diff -r package/ :.)` — specifically inspect `package.json` `scripts.preinstall` and `scripts.postinstall` fields for any additions not present in your last clean commit. Use `sha256sum` to verify the clean @bitwarden/cli tarball hash against Bitwarden's published release checksums on GitHub before deploying the replacement.

Evidence: Preserve BEFORE removing artifacts: (1) a forensic copy of the malicious @bitwarden/cli v2026.4.0 tarball from each artifact registry and package cache — hash it with `sha256sum` and store offline — this is the primary malware sample for later static analysis of the exfiltration payload and self-propagation logic; (2) complete contents of any downstream package versions your organization published after April 22 before republishing clean versions, including their `package.json`, all JS files, and the `.npmignore` — the self-propagation mechanism will have injected code into specific files that must be documented as evidence; (3) Artifactory or Nexus download audit logs showing which internal consumers pulled v2026.4.0 from your private registry, to extend the blast radius assessment beyond direct npm installs.

Step 4: Recovery — validate that CI/CD pipelines are pulling verified clean package versions by confirming SHA integrity hashes against Bitwarden's official release. Re-enable production deployments only after credential rotation is confirmed complete and downstream packages have been audited. Monitor cloud provider access logs and npm publish activity for at least 30 days post-remediation for signs of persistent access using credentials that may have been exfiltrated before detection.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST IR-4 (Incident Handling), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 8.2 (Collect Audit Logs)

Compensating: Enforce SHA integrity in `package-lock.json` by running `npm ci` (not `npm install`) in all CI/CD pipelines — `npm ci` enforces the exact integrity hash stored in the lockfile and fails if the resolved package hash does not match. Add a pre-deploy pipeline step that runs `npm audit --audit-level=high` and `node -e "const p=require('./node_modules/@bitwarden/cli/package.json'); if(p.version==='2026.4.0') process.exit(1)"` as a hard gate. For 30-day monitoring of cloud credential misuse without a SIEM, configure AWS CloudTrail to deliver logs to S3 and schedule a daily Lambda or cron job running `aws cloudtrail lookup-events` filtered by the newly issued IAM key IDs, alerting on any `GetSecretValue`, `AssumeRole`, or API calls from IP addresses outside known CI/CD runner egress ranges. Monitor npm publish activity for your organization's packages by subscribing to `https://registry.npmjs.org/-/npm/v1/hooks/package/@/` webhooks and alerting on any publish event not initiated by a known release pipeline identity.

Evidence: Capture at recovery validation stage: (1) `npm ci --dry-run` output from a clean pipeline run showing resolved integrity hashes for @bitwarden/cli matching Bitwarden's published SHA — retain this as the verified baseline artifact; (2) a snapshot of all newly issued AWS IAM access key IDs, GCP service account key IDs, and Azure service principal credentials with their creation timestamps — this establishes the clean credential baseline against which any future CloudTrail/Audit Log anomalies will be compared during the 30-day monitoring window; (3) npm publish logs from your organization's registry for all packages republished as part of eradication, confirming the new package tarball SHAs — retain these to evidence the clean republication if downstream consumers later report issues.

Step 5: Post-Incident — this attack exploited the absence of cryptographic integrity verification on npm package installs (CWE-494) and broad credential availability in CI/CD environments. Control improvements to implement: enforce npm package integrity verification with lockfile pinning and Subresource Integrity checks; adopt short-lived, scoped CI/CD credentials using OIDC-based authentication (e.g., GitHub Actions OIDC for AWS/Azure/GCP) instead of long-lived static keys; implement a private package mirror or registry proxy with allow-listing and integrity scanning; add software composition analysis (SCA) tooling to CI/CD pipelines to flag unexpected package version changes before installation.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST SA-15 (Development Process, Standards, and Tools), NIST SC-28 (Protection of Information at Rest), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access)

Compensating: For a 2-person team with no enterprise SCA budget: (1) implement `npm ci` with a `.npmrc` containing `audit=true` and `ignore-scripts=true` to block postinstall script execution entirely — this would have prevented the TeamPCP v2026.4.0 payload from executing at install time; (2) configure GitHub Actions OIDC for AWS by adding `permissions: id-token: write` to workflow YAML and using `aws-actions/configure-aws-credentials` with `role-to-assume` instead of storing `AWS_ACCESS_KEY_ID` secrets — this eliminates the static long-lived keys that v2026.4.0 targeted; (3) deploy Verdaccio (free, open-source) as a private npm proxy with an allowlist of approved packages and versions — configure `npmjs.uplink` with `maxage: 30m` to cache upstream packages and enable manual approval for version bumps on security-sensitive packages like @bitwarden/cli; (4) add a free OSSF Scorecard GitHub Action (`ossf/scorecard-action`) to all repos to surface supply chain risk signals including dependency pinning hygiene; (5) write a YARA rule targeting the known pattern of npm packages exfiltrating environment variables via `process.env` reads followed by HTTP POST — apply via `yara node_modules/@bitwarden/cli/` as a post-install CI step.

Evidence: Preserve for lessons-learned and potential regulatory disclosure: (1) timeline reconstruction document mapping the 90-minute availability window of v2026.4.0 (approx. 2026-04-22T00:00Z to 2026-04-22T01:30Z) against all CI/CD pipeline run timestamps and developer workstation npm install events — this is the definitive blast radius evidence; (2) complete list of all credentials confirmed rotated with pre/post rotation timestamps, organized by credential type (npm tokens, SSH keys, AWS key IDs, Azure SPN object IDs, GCP service account key IDs) — required for any regulatory breach notification determination; (3) static analysis report of the malicious v2026.4.0 tarball documenting the self-propagation mechanism's code, targeted environment variables, and exfiltration endpoint — this enables threat intelligence sharing with npm security team, GitHub Security Lab, and CISA if warranted.

Detection Guidance

Primary detection signals: (1) Package presence, search all package-lock.json, yarn.lock, and npm-shrinkwrap.json files across repositories and build systems for the string '@bitwarden/cli' with version '2026.4.0'. (2) CI/CD log review, examine GitHub Actions workflow run logs, Jenkins build logs, and equivalent pipeline logs for install events on April 22, 2026, specifically npm install or npx invocations that resolved to v2026.4.0. (3) Network egress, review proxy and firewall logs for outbound HTTPS POST requests to unfamiliar

domains originating from build agents during npm install sequences on April 22; exfiltration technique T1567.001 suggests web service destinations, potentially code repository infrastructure. (4) Cloud provider audit logs, query AWS CloudTrail for anomalous AssumeRole, GetSecretValue, or ListBuckets calls; Azure Monitor for service principal authentication from unexpected IPs; GCP Cloud Audit Logs for unusual API activity, all correlated to the April 22 window and originating systems. (5) Downstream package integrity, compare published package contents in your npm organization against known-good versions using checksums; look for unexpected additions to scripts.preinstall or scripts.postinstall fields in package.json of packages you own. Behavioral indicators: npm preinstall script execution accessing environment variables at scale (T1552.001), unexpected file reads of SSH key directories (~/.ssh/), and outbound connections from build processes to non-registry domains. IOC specifics are sourced from JFrog Security and Ox Security research; consult their published reports for confirmed command-and-control infrastructure details.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`See JFrog Security research post for confirmed package hash values`	SHA integrity hash for @bitwarden/cli v2026.4.0 malicious npm package — consult https://research.jfrog.com/post/bitwarden-cli-hijack/ for confirmed values; not reproduced here to avoid transcription error	MEDIUM
URL	`See JFrog Security and Ox Security reports for confirmed C2 and exfiltration endpoints`	Exfiltration and C2 infrastructure attributed to TeamPCP — IOC lists published in primary research; not reproduced here pending direct source verification	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1650** — Acquire Access
- **T1567** — Exfiltration Over Web Service
- **T1543** — Create or Modify System Process
- **T1567.001** — Exfiltration to Code Repository
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1027** — Obfuscated Files or Information
- **T1552.001** — Credentials In Files
- **T1078.004** — Cloud Accounts
- **T1059.007** — JavaScript
- **T1552.004** — Private Keys
- **T1554** — Compromise Host Software Binary

NIST-800-53R5

- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **IA-5** — Authenticator Management
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

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

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **5.2** — Use Unique Passwords
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **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

ISO-27001-2022

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1650	Acquire Access	Resource-Development
T1567	Exfiltration Over Web Service	Exfiltration
T1543	Create or Modify System Process	Persistence
T1567.001	Exfiltration to Code Repository	Exfiltration

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1027	Obfuscated Files or Information	Defense-Evasion
T1552.001	Credentials In Files	Credential-Access
T1078.004	Cloud Accounts	Defense-Evasion
T1059.007	JavaScript	Execution
T1552.004	Private Keys	Credential-Access
T1554	Compromise Host Software Binary	Persistence

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/bitwarden-cli-npm-pa...	T3
Bitwarden CLI Compromised: Inside the Shai-Hulud Supply Chain ...	https://www.ox.security/blog/shai-hulud-bitwarden-cli-supply-chain-...	T3
JFrog Security: Possible compromised Supply-Chain of Bitwarden CLI	https://github.com/bitwarden/clients/issues/20353	T3
TeamPCP Campaign Spreads to npm via a Hijacked Bitwarden CLI	https://research.jfrog.com/post/bitwarden-cli-hijack/	T3
Bitwarden CLI compromised - Vaultwarden affected? #7126 - GitHub	https://github.com/dani-garcia/vaultwarden/discussions/7126	T3

Generated 2026-04-24 06:45 UTC by TJS Security Command Center