INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-23 13:38 UTC

Bitwarden CLI Supply Chain Compromise: GitHub Actions Abuse Turns Developer Tools Into CI/CD Backdoors

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0207
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	@bitwarden/cli v2026.4.0 (npm); GitHub Actions trusted publishing pipeline; secondary exposure: Claude, Kiro, Cursor, Codex CLI, Aider AI coding tool configurations; CI/CD pipelines using affected npm token scope
Published	2026-04-23T09:42:00
Discovery Source	Rss

Executive Summary

On April 22, 2026, attackers compromised the Bitwarden command-line interface package on npm by hijacking its automated publishing workflow, inserting credential-harvesting code that ran silently in any environment that installed the affected version during a 90-minute window. Developers and CI/CD pipelines that pulled @bitwarden/cli@2026.4.0 during that window are at risk of cascading compromise across cloud infrastructure, source code repositories, and AI coding tool configurations. This is not a breach of Bitwarden vaults or end-user password data; the risk is concentrated in engineering and DevSecOps environments where the CLI is used programmatically.

Technical Analysis

Affected package: @bitwarden/cli@2026.4.0 (npm). Removed from registry approximately 90 minutes after publication on April 22, 2026. Attack vector: threat actors hijacked a GitHub Actions workflow within Bitwarden's CI/CD pipeline and exploited npm's trusted publishing mechanism, assessed as a rare or notable abuse of npm trusted publishing in a supply chain attack context. The malicious package contained embedded malicious code (CWE-506) that harvested GitHub tokens, npm tokens, cloud provider credentials, and shell history. Exfiltration occurred over encrypted channels to a domain impersonating Checkmarx and to attacker-controlled public GitHub repositories (CWE-829, CWE-312, CWE-522). Relevant MITRE techniques: T1195.001 (Supply Chain Compromise: Compromise Software Dependencies), T1528 (Steal Application Access Token), T1552.001

(Credentials In Files), T1552.004 (Private Keys), T1567.001 (Exfiltration to Code Repository), T1027 (Obfuscated Files or Information), T1078.004 (Valid Accounts: Cloud Accounts). No CVE assigned as of this writing. No vendor CVSS vector provided. Attribution unconfirmed. Organizations should treat any install of v2026.4.0 as a full credential compromise event and initiate immediate credential rotation.

Action Checklist

1. Step 1: Containment, Immediately identify any system, pipeline, or developer workstation that executed an npm install or npm ci referencing @bitwarden/cli@2026.4.0 between approximately 00:00 and 02:00 UTC on April 22, 2026 (confirm exact window from Bitwarden's advisory). Isolate affected CI/CD runners and immediately begin rotating all tokens and secrets accessible from those environments (GitHub PATs, npm tokens, cloud provider keys). Do not wait for confirmation of exfiltration to initiate rotation.
2. Step 2: Detection, Query npm audit logs, package-lock.json files, and CI/CD pipeline logs for installs of @bitwarden/cli version 2026.4.0. Search outbound network logs for connections to any domain impersonating 'Checkmarx' and for unexpected push activity to public GitHub repositories. Review shell history files on affected systems for unexpected commands. Check GitHub and npm token usage logs for anomalous API calls following the exposure window.
3. Step 3: Eradication, Rotate all credentials accessible from any affected environment: GitHub PATs and Actions secrets, npm tokens, cloud provider keys (AWS, GCP, Azure). Remove v2026.4.0 from any internal artifact mirrors or caches. Upgrade to the next clean release of @bitwarden/cli verified against Bitwarden's official advisory. Audit GitHub Actions workflow permissions and restrict GITHUB_TOKEN scopes to the minimum required.
4. Step 4: Recovery, After credential rotation, validate CI/CD pipeline integrity by re-running builds from a known-clean state. Confirm no unauthorized commits, package publishes, or infrastructure changes occurred during or after the exposure window. Monitor cloud provider billing and API usage for anomalies. Re-scan all environments that touched v2026.4.0 with a dependency integrity tool before restoring to production.
5. Step 5: Post-Incident, Implement npm package version pinning with integrity hash verification (package-lock.json integrity fields) across all pipelines. Evaluate adoption of npm provenance attestation to detect future trusted publishing abuse. Audit GitHub Actions workflow permissions organization-wide, applying least-privilege to GITHUB_TOKEN and restricting which workflows can trigger package publishes. Consider requiring manual approval gates for any workflow that publishes to npm or cloud artifact registries.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal counsel immediately if forensic review confirms that secrets accessible from affected runners included credentials to systems storing PII, PHI, or PCI-scoped data, or if evidence shows the attacker executed secondary publishes from your npm tokens — either condition triggers breach notification obligations under GDPR Article 33, HIPAA §164.410, or applicable state breach notification laws, and exceeds the response capability of the IR team alone.

Recovery Notes	Do not restore any CI/CD runner or developer environment to production until `npm audit signatures` passes clean against the replacement @bitwarden/cli version confirmed by Bitwarden's official advisory, and until all credentials accessible from those environments have completed rotation with confirmed invalidation of the prior credentials verified via each provider's token audit log. Monitor GitHub organization audit logs, AWS CloudTrail, and GCP Audit Logs continuously for a minimum of 30 days post-recovery for signs of persistence — specifically watch for new OAuth app authorizations, unexpected IAM role assumption patterns, and any npm publish events from your organization's packages, as attackers who obtained npm tokens during the exposure window may have staged a delayed secondary supply chain attack. Treat any AI coding tool configuration file (`.cursor/`, `.aider.conf`, Kiro settings) on affected developer workstations as potentially modified and restore from a pre-compromise backup or clean re-installation before those tools are used against production codebases.
Forensic Artifacts	npm cache tarball for @bitwarden/cli@2026.4.0 at ~/.npm/_cacache/content-v2/ (Linux) or %AppData%\npm-cache (Windows) — contains the malicious postinstall script that executed credential harvesting; extract and hash (SHA-256) before any cache-clearing operations | GitHub Actions workflow run logs for all runs between 2026-04-22T00:00:00Z and 2026-04-22T02:00:00Z — downloadable via `gh run download ` — preserves evidence of which secrets (GITHUB_TOKEN, repository secrets) were injected into the runner environment where the malicious npm postinstall hook executed | Outbound network connection logs (DNS queries and TCP session records) from CI runners and developer workstations during the exposure window — specifically DNS lookups and HTTPS POST requests to Checkmarx-impersonating domains, which is the identified exfiltration channel for harvested environment variable contents including credential strings | Environment variable state at time of npm install — recoverable from /proc//environ (Linux) snapshots if the runner was captured live, or from CI/CD platform secret audit logs (GitHub Actions: Settings → Secrets → usage log) — documents precisely which API keys, cloud credentials, and tokens were in scope for exfiltration by the postinstall hook | Git reflog and commit history across all repositories accessible to the compromised GitHub PATs or GITHUB_TOKEN — run `git reflog --all` and `git log --all --format='%H %ae %ai %s' --since=2026-04-22` to detect unauthorized commits, branch creations, or tag modifications that could indicate attacker persistence or secondary supply chain staging using your own repositories

Per-Action IR Details

Step 1: Containment — Immediately identify any system, pipeline, or developer workstation that executed an npm install or npm ci referencing @bitwarden/cli@2026.4.0 between approximately 00:00 and 02:00 UTC on April 22, 2026 (confirm exact window from Bitwarden's advisory). Isolate affected CI/CD runners and revoke all tokens and secrets accessible from those environments without waiting for confirmation of exfiltration.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST AC-17 (Remote Access), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: For teams without a centralized SIEM: run `grep -r '@bitwarden/cli' /path/to/repos --include='package-lock.json'` across all developer machines and CI runner file systems to locate version 2026.4.0 references. On GitHub Actions runners, query workflow run logs via GitHub CLI: `gh run list --json startedAt,name,conclusion | jq '.[] | select(.startedAt >= "2026-04-22T00:00:00Z" and .startedAt ` for every token scoped to the affected pipeline. Isolate CI runners by disabling the runner in GitHub Actions settings (Settings → Actions → Runners → set to Offline) or by blocking outbound network access with a host-based firewall rule (`ufw deny

out` on Linux runners).

Evidence: Before isolating runners, capture: (1) a full snapshot of the runner's npm cache directory (`~/.npm/_cacache/` on Linux, `%AppData%\npm-cache` on Windows) to preserve the v2026.4.0 package tarball and its embedded malicious postinstall script; (2) the process tree at time of discovery using `ps auxf` (Linux) or `Get-WmiObject Win32_Process` (Windows) to identify any child processes spawned by the npm postinstall hook from @bitwarden/cli@2026.4.0; (3) `/proc/net/tcp` or `netstat -antp` output to capture any live outbound connections to attacker-controlled domains at moment of isolation; (4) environment variable dumps (`/proc//environ` on Linux) from the npm install process to document which secrets (GITHUB_TOKEN, AWS credentials, npm tokens) were accessible in the runner's environment at time of compromise.

Step 2: Detection — Query npm audit logs, package-lock.json files, and CI/CD pipeline logs for installs of @bitwarden/cli version 2026.4.0. Search outbound network logs for connections to any domain impersonating 'Checkmarx' and for unexpected push activity to public GitHub repositories. Review shell history files on affected systems for unexpected commands. Check GitHub and npm token usage logs for anomalous API calls following the exposure window.

NIST Phase: Detection Analysis

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

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

Compensating: Without SIEM: (1) Search all package-lock.json files recursively: `find / -name 'package-lock.json' 2>/dev/null | xargs grep -l '"@bitwarden/cli"' | xargs grep '2026.4.0'`. (2) Query npm access log at `~/.npm/_logs/` for install timestamps matching the 00:00–02:00 UTC window. (3) Search DNS query logs or use `journalctl -u systemd-resolved` (Linux) for queries to Checkmarx-impersonating domains — look for typosquatted variants such as `checkmarx[.]io`, `chekmarx[.]com`, `checkmar[.]xyz`. (4) Run `git log --all --oneline --since='2026-04-22T00:00:00' --until='2026-04-22T02:00:00'` on all local repos to detect unauthorized commits. (5) Query GitHub audit log via API: `GET /orgs/{org}/audit-log?phrase=action:git.push&after=2026-04-22T00:00:00Z` and `GET /orgs/{org}/audit-log?phrase=action:npm.publish`. (6) Use osquery to search shell history: `SELECT * FROM shell_history WHERE command LIKE '%bitwarden%' OR command LIKE '%2026.4.0%'`.

Evidence: Capture before pivoting on detections: (1) the malicious postinstall script embedded in the v2026.4.0 package (extract from npm cache at `~/.npm/_cacache/content-v2/` by locating the tarball for `@bitwarden/cli@2026.4.0` and inspecting `package/scripts/postinstall.js` or equivalent); (2) outbound HTTP/DNS logs filtered for the 00:00–02:00 UTC window showing any beacon or exfiltration to non-Bitwarden infrastructure — specifically look for base64-encoded POST bodies or multipart uploads to unknown endpoints that would carry harvested environment variables; (3) GitHub Actions workflow run logs (downloadable via `gh run download `) for the affected window, preserving the raw log before GitHub's 90-day retention purges them; (4) npm token last-used timestamps from `npm token list` or the npm registry API to identify tokens that authenticated after the install window from unexpected IP ranges; (5) `.bash_history` / `.zsh_history` and PowerShell `ConsoleHost_history.txt` (`%APPDATA%\Microsoft\Windows\PowerShell\PSReadLine\`) from all affected developer workstations.

Step 3: Eradication — Rotate all credentials accessible from any affected environment: GitHub PATs and Actions secrets, npm tokens, cloud provider keys (AWS, GCP, Azure), and API keys for AI coding tools (Claude, Kiro, Cursor, Codex CLI, Aider). Remove v2026.4.0 from any internal artifact mirrors or caches. Upgrade to the next clean release of @bitwarden/cli verified against Bitwarden's official advisory. Audit GitHub Actions workflow permissions and restrict GITHUB_TOKEN scopes.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST AC-2 (Account Management), NIST CM-7 (Least Functionality), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Credential rotation without enterprise PAM: (1) AWS — `aws iam delete-access-key --access-key-id ` then `aws iam create-access-key`; audit CloudTrail for API calls made by the compromised key using `aws cloudtrail lookup-events --lookup-attributes AttributeKey=AccessKeyId,AttributeValue=`. (2) GCP — `gcloud iam service-accounts keys delete --iam-account=`. (3) Azure — revoke via `az ad app credential delete`. (4) For AI tool API keys (Claude/Anthropic, OpenAI Codex): log into each provider console and revoke keys associated with affected developer accounts — document each revocation with timestamp for audit trail. (5) Remove v2026.4.0 from internal mirrors: `npm unpublish @bitwarden/cli@2026.4.0 --registry ` or delete from Artifactory/Nexus via their REST APIs. (6) Restrict GITHUB_TOKEN: add `permissions: read-all` at workflow top level and explicitly grant `contents: write` only to the specific job that requires it, using GitHub Actions workflow YAML `permissions` key.

Evidence: Before rotating credentials, preserve: (1) AWS CloudTrail events for the compromised access key covering 2026-04-22T00:00:00Z through time of rotation — export via `aws cloudtrail lookup-events` filtered by the key ID, preserving evidence of any S3 exfiltration, IAM privilege escalation, or Lambda/EC2 modifications; (2) GitHub audit log entries for each PAT being rotated, specifically capturing any `git.push`, `repository.create`, `org.invite_member`, or `workflow.created` events post-compromise window; (3) npm registry access logs for the compromised npm token showing any `npm publish` calls, which would indicate the attacker attempted secondary supply chain propagation through packages your pipeline publishes; (4) AI tool API usage logs (Anthropic Console, OpenAI usage dashboard) for the affected key covering the exposure window — an attacker with a Claude/Codex API key could have exfiltrated code context or used the key for unauthorized inference.

Step 4: Recovery — After credential rotation, validate CI/CD pipeline integrity by re-running builds from a known-clean state. Confirm no unauthorized commits, package publishes, or infrastructure changes occurred during or after the exposure window. Monitor cloud provider billing and API usage for anomalies. Re-scan all environments that touched v2026.4.0 with a dependency integrity tool before restoring to production.

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 IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.3 (Perform Automated Operating System Patch Management)

Compensating: For teams without enterprise integrity monitoring: (1) Verify clean build state by running `npm ci --ignore-scripts` on a freshly provisioned runner (disable postinstall hooks to prevent re-execution of any cached malicious scripts) and then run `npm audit signatures` to verify package registry signatures for all installed packages. (2) Validate pipeline output integrity: compare SHA-256 hashes of build artifacts produced pre- and post-compromise using `sha256sum` (Linux) or `Get-FileHash` (PowerShell). (3) Audit all git repositories touched by the affected pipeline for unauthorized commits: `git log --all --format='%H %ae %ai %s' --since='2026-04-22T00:00:00' --until=` — any commit author email not matching your team roster warrants investigation. (4) Check cloud billing anomalies: enable AWS Cost Explorer anomaly detection alerts or GCP Billing Budget alerts; for immediate review, use `aws ce get-cost-and-usage` for the 2026-04-22 period. (5) Re-scan with `npm audit` and `node_modules/.bin/better-npm-audit` after upgrading to the verified clean @bitwarden/cli release confirmed in Bitwarden's official advisory.

Evidence: Capture for recovery validation: (1) signed provenance attestation for the replacement @bitwarden/cli package, obtainable via `npm audit signatures` — verify the attestation chain traces to Bitwarden's official npm trusted publisher configuration, not the compromised GitHub Actions workflow; (2) diff of all infrastructure-as-code (Terraform state files, CloudFormation stacks, GitHub Actions workflow YAML) between pre-compromise baseline and post-recovery state to detect any backdoor persistence mechanisms the attacker may have inserted during the exposure window; (3) cloud provider API call history for the 24 hours following the exposure window — specifically look for IAM role modifications, new access key creation, Lambda function updates, or S3 bucket policy changes that would indicate attacker persistence beyond the initial token theft.

Step 5: Post-Incident — Implement npm package version pinning with integrity hash verification (package-lock.json integrity fields) across all pipelines. Evaluate adoption of npm provenance attestation to detect future trusted publishing abuse. Audit GitHub Actions workflow permissions organization-wide,

applying least-privilege to GITHUB_TOKEN and restricting which workflows can trigger package publishes. Consider requiring manual approval gates for any workflow that publishes to npm or cloud artifact registries.

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 CM-7 (Least Functionality), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Free controls to implement immediately: (1) Enforce package-lock.json integrity — add `npm ci` (not `npm install`) to all pipeline definitions; `npm ci` validates the `integrity` field (SHA-512 hash) in package-lock.json against the downloaded tarball, which would have flagged the tampered v2026.4.0. (2) Enable npm provenance attestation verification: add `npm audit signatures` as a mandatory pipeline step; this verifies each package was built and published by a trusted CI system using OIDC, detecting the kind of GitHub Actions workflow hijack used in this campaign. (3) Add a Sigma rule to your log pipeline (or run as a cron-based grep): detect any npm install of packages with `postinstall` scripts that make outbound network connections — use `sysmon` Event ID 3 (Network Connection) with parent process `node.exe` or `npm.cmd` as the detection anchor. (4) Implement GitHub Actions environment protection rules (free on all GitHub plans): require manual reviewer approval for any job that has `contents: write` permission or publishes to npm, enforced via `environment: production` with required reviewers set. (5) Pin all GitHub Actions to commit SHA rather than mutable tags (e.g., `uses: actions/checkout@11bd71901bbe5b1630ceea73d27597364c9af68` instead of `@v4`) to prevent tag-hijacking attacks analogous to this trusted publishing compromise.

Evidence: Document for lessons-learned report and future detection: (1) the exact package-lock.json integrity hash discrepancy between the signed clean release and v2026.4.0 — this becomes the baseline IOC for YARA rules targeting future npm supply chain attacks against your package set; (2) the GitHub Actions workflow diff showing the injected malicious publishing step, preserved as evidence for the incident record and as a template for detection rules monitoring workflow YAML changes via `git diff` in a pre-commit hook or GitHub's `workflow_dispatch` audit events; (3) a complete inventory of all pipelines, repositories, and developer workstations confirmed exposed, mapped to the credential types accessible from each — this scoping document satisfies NIST IR-6 (Incident Reporting) requirements and supports any regulatory breach notification assessment if PII or PHI was accessible from the compromised environments.

Detection Guidance

Note: Specific indicators of compromise (Checkmarx-impersonating domain, exfiltration server IPs) are not yet public as of this writing. Primary detection should focus on package version logs, GitHub token anomalies, and file system indicators until IOCs are published by Bitwarden or threat intelligence providers. Detection targets: (1) Package install records, search package-lock.json, yarn.lock, and CI/CD build logs for '@bitwarden/cli' pinned to '2026.4.0'. (2) Network egress, query firewall and proxy logs for outbound connections to any domain containing 'checkmarx' that is not the legitimate checkmarx.com, and for unexpected HTTPS POSTs to github.com/repos from CI runner IPs during the exposure window. (3) Token usage anomalies, pull GitHub audit logs (Organization > Security log) and filter for oauth_access.create, personal_access_token.create, or unexpected repo.create events in the April 22 window. Check npm access logs for token usage outside normal pipeline hours. (4) File system indicators, on any system that ran the package, inspect ~/.bash_history, ~/.zsh_history, and AI tool config directories for unexpected reads or copies. (5) Behavioral indicators, unexpected cron jobs (T1053.003) or shell scripts (T1059.004) created on CI runners following the install.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`Checkmarx-impersonating exfiltration domain — specific domain not yet publicly confirmed`	Threat actors used a domain impersonating Checkmarx (checkmarx.com) to receive encrypted stolen credentials. Specific domain pending publication by Bitwarden or threat intelligence sources.	LOW
URL	`Public GitHub repository used for credential exfiltration — specific repository not yet publicly identified`	Stolen data was exfiltrated to one or more attacker-controlled public GitHub repositories. Specific URL pending disclosure.	LOW
HASH	`@bitwarden/cli@2026.4.0 npm package — integrity hash pending vendor publication`	The malicious npm package version 2026.4.0. Compare package-lock.json integrity field against Bitwarden's official advisory once published.	HIGH

Framework Mappings

MITRE-ATTACK

- **T1567.001** — Exfiltration to Code Repository
- **T1078.004** — Cloud Accounts
- **T1027** — Obfuscated Files or Information
- **T1053.003** — Cron
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1078.001** — Default Accounts
- **T1528** — Steal Application Access Token
- **T1608.001** — Upload Malware
- **T1552.001** — Credentials In Files
- **T1552.004** — Private Keys
- **T1059.004** — Unix Shell

NIST-800-53R5

- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **IA-5** — Authenticator Management
- **SR-2** — Supply Chain Risk Management Plan
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

- **A04:2021** — Insecure Design

- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **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
- **164.312(e)(1)** — Transmission Security

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.8.24** — Use of cryptography
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1567.001	Exfiltration to Code Repository	Exfiltration
T1078.004	Cloud Accounts	Defense-Evasion
T1027	Obfuscated Files or Information	Defense-Evasion
T1053.003	Cron	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1078.001	Default Accounts	Defense-Evasion
T1528	Steal Application Access Token	Credential-Access
T1608.001	Upload Malware	Resource-Development
T1552.001	Credentials In Files	Credential-Access
T1552.004	Private Keys	Credential-Access

Technique ID	Technique Name	Tactic
T1059.004	Unix Shell	Execution

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/04/bitwarden-cli-compromised-in-ongo...	T3
bitwarden/cli: The command line vault (Windows, macOS, & Linux).	https://github.com/bitwarden/cli	T3
9 AI Coding Alternatives for Terminal Development: Complete 2026 ...	https://pasqualepillitteri.it/en/news/386/ai-coding-cli-alternative...	T3
I Scanned 9 Popular AI Coding Tools for Security Issues. Here's ...	https://dev.to/ferg-cod3s/i-scanned-9-popular-ai-coding-tools-for-s...	T3
hermit-packages/bitwarden-cli.hcl at master - GitHub	https://github.com/cashapp/hermit-packages/blob/master/bitwarden-cl...	T3

Generated 2026-04-23 13:38 UTC by TJS Security Command Center