INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-02 18:36 UTC

Shai-Hulud Campaign Enters Third Generation: SAP and Bitwarden Ecosystems Hit in Coordinated April 2026 npm Supply Chain Offensive

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0261
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm registry; @bitwarden/cli (typosquat/malicious package); @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service (SAP CAP ecosystem, 570,000+ combined weekly downloads); mbt (SAP MTA build tool); Checkmarx KICS Docker images; Checkmarx GitHub Actions (ast-github-action); Checkmarx VS Code extensions (ast-results, cx-dev-assist); GitHub Actions CI/CD pipelines; AWS, Azure Key Vault, GCP Secret Manager; Kubernetes; Electrum wallets
Published	2026-05-02T00:10:33+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

A threat actor tracked as TeamPCP is running a third-generation npm supply chain campaign, placing malicious packages that impersonate widely used enterprise tools including SAP Cloud Application Programming (CAP) ecosystem libraries and the Bitwarden CLI password manager. Organizations consuming these packages via automated CI/CD pipelines are at risk of credential theft across AWS, Azure, GCP, and Kubernetes environments without any user interaction beyond a routine dependency install. The business risk is direct exfiltration of cloud access keys, pipeline secrets, and cryptocurrency wallets from enterprise build infrastructure at scale.

Technical Analysis

TeamPCP's Generation 3 campaign deploys malicious npm packages mimicking @bitwarden/cli and four SAP CAP packages (@cap-js/sqlite, @cap-js/postgres, @cap-js/db-service, and mbt). Combined, the SAP packages carry 570,000+ weekly downloads, providing high-volume pipeline access. Checkmarx tooling (KICS Docker images, ast-github-action, VS Code extensions ast-results and cx-dev-axis) was also implicated. The payload

chain uses a custom obfuscation layer (CWE-506), a Bun JavaScript runtime bootstrapper for execution (T1059.007), and a GitHub commit dead-drop for C2 (T1102.001), all three markers consistent across Generations 1, 2, and 3. Post-install scripts exfiltrate: AWS, Azure Key Vault, and GCP Secret Manager secrets (T1552.001, T1552.004); Kubernetes cluster credentials (T1078.004); GitHub Actions CI/CD environment secrets (T1195.001); and Electrum wallet data (T1496). Exfiltration occurs via automated data transfer (T1020) with no user interaction required beyond dependency resolution. The campaign also uses masquerading (T1036.005) and persistence mechanisms (T1543, T1554). Attribution to TeamPCP is supported by identical obfuscation signatures, Bun bootstrapper reuse, and the GitHub dead-drop C2 pattern documented in Microsoft's December 2025 Shai-Hulud 2.0 advisory (high confidence). Relevant CWEs: CWE-312 (cleartext storage), CWE-829 (inclusion of untrusted functionality), CWE-506 (embedded malicious code), CWE-798 (hard-coded credentials), CWE-494 (download without integrity check). No CVE is assigned. No vendor patch exists for the malicious packages, remediation is removal and registry audit.

Action Checklist

1. Step 1: Containment. Immediately audit your npm dependency tree for @bitwarden/cli, @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service, mbt, Checkmarx ast-github-action, ast-results, and cx-dev-axis. Cross-reference installed versions against the malicious package hashes published by Wiz Threat Research and Unit 42. Suspend CI/CD pipelines that consume any flagged package until cleared. Revoke all cloud credentials (AWS IAM keys, Azure Key Vault access tokens, GCP service account keys) accessible from affected build environments.
2. Step 2: Detection. Search pipeline logs and npm install logs for installation events involving the flagged packages. Hunt for outbound DNS or HTTPS requests to GitHub raw content endpoints (github.com/raw or raw.githubusercontent.com) originating from build agents or developer workstations, this is the dead-drop C2 channel (T1102.001). Check for unexpected Bun runtime process execution in CI/CD runner logs. Review AWS CloudTrail, Azure Monitor, and GCP Cloud Audit Logs for anomalous secret access or IAM key usage originating from build infrastructure. Inspect Kubernetes audit logs for credential enumeration activity.
3. Step 3: Eradication. Remove all malicious or suspect package versions from your environment and dependency lock files. Pin dependencies to verified, integrity-checked versions using npm's --ignore-scripts flag during installs where feasible to block post-install payload execution. Rotate all cloud secrets, API keys, and Kubernetes credentials that were accessible from any affected pipeline or developer machine. Remove any Checkmarx tooling versions flagged by Wiz/Unit 42 IOCs and re-deploy from verified sources. If Electrum was present on affected machines, treat wallet keys as compromised.
4. Step 4: Recovery. After credential rotation, validate that no new API calls are being made with the old credentials (monitor CloudTrail, Azure Monitor, GCP Audit Logs for 72 hours post-rotation). Re-run dependency installs from a clean, isolated environment against a verified npm mirror or private registry with integrity checks enforced. Confirm CI/CD pipeline secrets have been regenerated in GitHub Actions, not just rotated at the cloud provider. Validate Checkmarx tooling reinstalled from official vendor sources before re-enabling security scanning pipelines.
5. Step 5: Post-Incident. This campaign succeeds because most organizations lack dependency integrity verification and grant CI/CD pipelines overly broad secret access. Implement npm provenance attestation and Sigstore-based package verification for all pipeline dependencies. Scope CI/CD pipeline secrets to the minimum necessary environments and jobs (principle of least privilege). Deploy a private npm registry or allow-list mirror to gate package ingestion. Add behavioral detection rules for Bun runtime execution in

build environments and outbound GitHub raw content requests from pipeline agents. Review the Microsoft Shai-Hulud 2.0 advisory (December 2025) for Generation 2 indicators that may still persist in your environment.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and cloud provider security teams immediately if AWS CloudTrail, Azure Monitor, or GCP Audit Logs confirm that the old credentials were used after rotation (indicating active adversary persistence), if any Kubernetes cluster secrets were enumerated (indicating potential lateral movement beyond the build environment), or if PII or regulated data was accessible from the compromised build environment secrets (triggering breach notification obligations under GDPR Article 33, CCPA, or applicable state law).
Recovery Notes	After credential rotation, maintain active monitoring of all rotated credential identifiers in CloudTrail, Azure Monitor, and GCP Audit Logs for a minimum of 72 hours to detect any adversary still holding exfiltrated keys; any usage of revoked credentials during this window elevates the incident to an active breach requiring re-containment. Before re-enabling CI/CD pipelines, validate every dependency in the restored `package-lock.json` against npm provenance attestation records using `npm audit signatures` and confirm no transitive dependency resolves to a TeamPCP-associated package hash. Monitor build agent egress traffic for outbound requests to `raw.githubusercontent.com` for 30 days post-recovery, as the TeamPCP T1102.001 dead-drop pattern may indicate a previously undetected implant that survived eradication.
Forensic Artifacts	npm cache tarballs at `~/.npm/_cacache/` (Linux/macOS) or `%APPDATA%\npm-cache\` (Windows) — the malicious TeamPCP package tarballs are retained here after install and contain the actual postinstall payload scripts for malware analysis and hash verification against Wiz/Unit 42 IOCs | CI/CD runner process execution logs showing `bun` process spawned as a child of `npm` or `node` during `postinstall` hook execution — on Linux runners, captured via `/var/log/audit/audit.log` EXECVE records; on Windows runners via Sysmon Event ID 1 filtered on `ParentCommandLine contains npm` | AWS CloudTrail `GetSecretValue`, `ListSecrets`, `AssumeRole`, and `GetCallerIdentity` events with `sourceIPAddress` matching CI build agent egress IPs and `eventTime` correlating to `npm install` execution windows — direct evidence of TeamPCP credential harvester calling the AWS Secrets Manager API | Kubernetes API server audit log entries at `/var/log/kubernetes/audit.log` for `verb=get` and `verb=list` on `resource=secrets` from service account tokens associated with the build namespace — indicates the malware pivoted from the npm postinstall context into the cluster using injected credentials | Network capture or proxy logs showing HTTP GET requests to `https://raw.githubusercontent.com//` from build agents during or immediately after `npm install` execution — this is the T1102.001 web service dead-drop used by TeamPCP to deliver the second-stage payload or exfiltrate credentials, and the specific URI path will identify the campaign generation and C2 repository

Per-Action IR Details

Step 1: Containment — Immediately audit your npm dependency tree for @bitwarden/cli, @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service, mbt, Checkmarx ast-github-action, ast-results, and cx-dev-assist. Cross-reference installed versions against the malicious package hashes published by Wiz Threat Research and Unit 42. Suspend CI/CD pipelines that consume any flagged package until cleared. Revoke all cloud credentials (AWS IAM keys, Azure Key Vault access tokens, GCP service account keys) accessible from

affected build environments.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-3 (Configuration Change Control), NIST AC-2 (Account Management), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 5.1 (Establish and Maintain an Inventory of Accounts)

Compensating: Run `npm list --all --json 2>/dev/null | python3 -c "import sys,json; pkgs=json.load(sys.stdin); [print(k,v) for k,v in pkgs.get('dependencies',{}).items()]"` recursively to enumerate the full dependency tree, then grep for the exact package names. Cross-check installed package integrity with `npm pack @ --dry-run` and compare SHA-512 checksums against the Wiz/Unit 42 published hashes using `sha512sum`. To suspend pipelines without an enterprise orchestration platform, set a repository-level branch protection rule requiring a named reviewer or toggle GitHub Actions workflow runs to `on: workflow_dispatch` only. Revoke AWS IAM keys immediately via `aws iam delete-access-key --access-key-id ` and GCP service account keys via `gcloud iam service-accounts keys delete --iam-account=`.

Evidence: Before revoking credentials, capture a full snapshot of: (1) `package-lock.json` and `yarn.lock` files from every affected repository showing the resolved version and integrity hash for the flagged TeamPCP packages; (2) the `.npm/_cacache` directory on build agents, which retains tarball content for installed packages and can confirm malicious payload presence; (3) a `pip freeze`-style output via `npm list --all --json` before any remediation to preserve the dependency graph as forensic evidence; (4) AWS CloudTrail `GetSecretValue` and `AssumeRole` events, Azure Key Vault `SecretGet` audit entries, and GCP Cloud Audit `projects.secrets.versions.access` log events generated within the window of suspected malicious package execution — these confirm whether the TeamPCP credential harvester successfully exfiltrated secrets before containment.

Step 2: Detection — Search pipeline logs and npm install logs for installation events involving the flagged packages. Hunt for outbound DNS or HTTPS requests to GitHub raw content endpoints (github.com/raw or raw.githubusercontent.com) originating from build agents or developer workstations — this is the dead-drop C2 channel (T1102.001). Check for unexpected Bun runtime process execution in CI/CD runner logs. Review AWS CloudTrail, Azure Monitor, and GCP Cloud Audit Logs for anomalous secret access or IAM key usage originating from build infrastructure. Inspect Kubernetes audit logs for credential enumeration activity.

NIST Phase: Detection Analysis

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

Controls: NIST IR-5 (Incident Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Query GitHub Actions runner logs directly in `.github/workflows/` run history for `npm install` stdout containing the flagged package names and their resolved versions. For Bun runtime detection on Linux CI runners, run `find /proc/*/exe -ls 2>/dev/null | grep bun` or search runner audit logs with `grep -r 'bun' /var/log/` and `journalctl | grep bun`. Deploy a Sigma rule targeting `CommandLine contains 'bun' AND ParentImage contains 'node'` using Sysmon Event ID 1 (Process Create) on Windows build agents. For network-layer C2 detection without a SIEM, use `tcpdump -i any -w /tmp/ci_capture.pcap 'host raw.githubusercontent.com'` during a controlled re-run of a suspect pipeline, then analyze the pcap with Wireshark filtering on `http.host contains "raw.githubusercontent.com"` to surface the T1102.001 dead-drop GET requests characteristic of this TeamPCP campaign. For Kubernetes, run `kubectl get events --all-namespaces | grep -i 'secret\|serviceaccount'` and review `kubectl logs -n kube-system` for anomalous API server requests.

Evidence: Capture before any log rotation or pipeline teardown: (1) Full GitHub Actions workflow run logs (`.github` run artifacts) showing `npm install` output and any `postinstall` script execution — the TeamPCP malicious packages trigger credential harvesting via `postinstall` hooks, so look for script execution immediately following package resolution; (2) DNS resolver logs or `/etc/hosts` query caches on build agents for lookups to `raw.githubusercontent.com` that do not correspond to any approved workflow step (MITRE T1102.001 dead-drop pattern); (3) Process execution logs from CI runner hosts: on Linux, `/var/log/audit/audit.log` for `execve` syscalls spawning `bun` processes; on Windows runners, Sysmon Event ID 1 for `bun.exe` spawned as a child of `node.exe` or `npm.cmd`; (4) AWS CloudTrail `LookupEvents` filtered on `eventName=GetSecretValue` and

`eventName=ListSecrets` with `sourceIPAddress` matching the build agent's egress IP — TeamPCP's harvester specifically targets secrets manager APIs; (5) Kubernetes API server audit log entries for `get secrets` and `list serviceaccounts` verbs from pod service account tokens, which would indicate the malware pivoted from the build environment into the cluster.

Step 3: Eradication — Remove all malicious or suspect package versions from your environment and dependency lock files. Pin dependencies to verified, integrity-checked versions using npm's --ignore-scripts flag during installs where feasible to block post-install payload execution. Rotate all cloud secrets, API keys, and Kubernetes credentials that were accessible from any affected pipeline or developer machine. Remove any Checkmarx tooling versions flagged by Wiz/Unit 42 IOCs and re-deploy from verified sources. If Electrum was present on affected machines, treat wallet keys as compromised.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST CM-3 (Configuration Change Control), NIST AC-2 (Account Management), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Remove flagged packages from lock files by running `npm uninstall --save` followed by `rm -rf node_modules package-lock.json` and a clean `npm install --ignore-scripts` to reinstall without executing any `postinstall` hooks — this directly blocks the TeamPCP payload execution mechanism. Clear the npm cache with `npm cache clean --force` to remove any cached malicious tarballs from `.npm/_cacache`. For Kubernetes credential rotation without enterprise tooling: `kubectl create secret generic --from-literal=...` then update all referencing manifests, followed by `kubectl delete secret `. For Electrum on compromised machines, transfer any remaining funds to a new wallet generated on a clean, air-gapped system before deleting `~/.electrum/wallets/` — the TeamPCP campaign specifically targets Electrum wallet files, so assume private keys are exfiltrated if the package was installed. For Checkmarx VS Code extensions (`ast-results`, `cx-dev-assist`), remove via `code --uninstall-extension checkmarx.` and validate removal with `code --list-extensions`.

Evidence: Before removing any artifacts, preserve: (1) A full disk image or at minimum a tar archive of the `node_modules` directory containing the malicious TeamPCP packages, captured with `tar -czf /evidence/node_modules_$(hostname)_$(date +%s).tar.gz ./node_modules` — this preserves the actual payload binary for malware analysis; (2) The Bun runtime binary if dropped to disk by the malicious postinstall script (commonly staged to `/tmp/`, `~/.bun/bin/bun`, or `%APPDATA%\bun\` on Windows) — hash it with `sha256sum` before deletion; (3) Any Electrum wallet files at `~/.electrum/wallets/` or `%APPDATA%\Electrum\wallets\` — these confirm whether wallet compromise is theoretical or evidenced; (4) Checkmarx extension files from VS Code's extension directory (`~/.vscode/extensions/checkmarx.*` or `%USERPROFILE%\.vscode\extensions\checkmarx.*`) for the flagged versions to provide to Wiz/Unit 42 for IOC validation; (5) A snapshot of currently active cloud credentials via `aws iam list-access-keys`, `gcloud iam service-accounts keys list`, and `az keyvault secret list` before rotation, establishing a clear pre/post rotation baseline for audit.

Step 4: Recovery — After credential rotation, validate that no new API calls are being made with the old credentials (monitor CloudTrail, Azure Monitor, GCP Audit Logs for 72 hours post-rotation). Re-run dependency installs from a clean, isolated environment against a verified npm mirror or private registry with integrity checks enforced. Confirm CI/CD pipeline secrets have been regenerated in GitHub Actions, not just rotated at the cloud provider. Validate Checkmarx tooling reinstalled from official vendor sources before re-enabling security scanning pipelines.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: For 72-hour post-rotation credential monitoring without a SIEM, set up AWS CloudTrail metric filters using the AWS CLI: `aws cloudwatch put-metric-filter --log-group-name CloudTrail/DefaultLogGroup --filter-name

OldKeyUsage --filter-pattern '{$.userIdentity.accessKeyId = ""}' --metric-transformations metricName=OldKeyHits,metricNamespace=SecurityMonitor,metricValue=1` — this creates an alert if the revoked key is attempted anywhere. For GCP, use `gcloud logging read 'protoPayload.authenticationInfo.serviceAccountKeyName="" --freshness=72h`. Validate GitHub Actions secrets were regenerated (not just updated at the provider) by reviewing `Settings > Secrets and variables > Actions` and confirming secret `updated_at` timestamps post-incident; old secrets that remain in GitHub Actions env context will be passed to runners even if the underlying cloud credential is rotated. For Checkmarx reinstallation integrity, verify the VS Code extension VSIX against the SHA-256 hash published on the Checkmarx official marketplace listing before installation.

Evidence: During recovery validation, capture and retain: (1) AWS CloudTrail `LookupEvents` output filtered on `accessKeyId=` for the 72-hour monitoring window — any hits indicate an active adversary still holds the old credential and the incident scope must be re-evaluated; (2) `npm install --ignore-scripts --dry-run` output from the clean recovery environment showing resolved package versions and integrity hashes against the new private registry or verified mirror, confirming no TeamPCP package re-introduced via transitive dependency; (3) GitHub Actions workflow run logs from the first clean pipeline execution post-recovery, showing successful secret injection from regenerated secrets and absence of outbound requests to `raw.githubusercontent.com`; (4) Checkmarx extension manifest file (`package.json` within the installed extension directory) showing the verified publisher, version, and integrity hash matching official Checkmarx release notes.

Step 5: Post-Incident — This campaign exploits the absence of dependency integrity verification and overly permissive CI/CD secret scoping. Implement npm provenance attestation and Sigstore-based package verification for all pipeline dependencies. Scope CI/CD pipeline secrets to the minimum necessary environments and jobs (principle of least privilege). Deploy a private npm registry or allow-list mirror to gate package ingestion. Add behavioral detection rules for Bun runtime execution in build environments and outbound GitHub raw content requests from pipeline agents. Review the Microsoft Shai-Hulud 2.0 advisory (December 2025) for Generation 2 indicators that may still persist in your environment.

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-15 (Development Process, Standards, and Tools), NIST CM-3 (Configuration Change Control), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Implement npm provenance enforcement at zero cost by adding `"publishConfig": {"provenance": true}` to `package.json` for all internal packages and enabling `npm audit signatures` in CI pipelines to verify Sigstore-backed package attestations for all external dependencies. For private registry deployment without enterprise budget, self-host Verdaccio (`npx verdaccio`) as an npm mirror with an allow-list configured in `config.yaml` under `packages:` — block all packages not explicitly listed to prevent TeamPCP-style typosquats from reaching build environments. Write a Sigma rule for Bun runtime detection in CI: `title: Bun Runtime Execution in CI Environment; logsource: category:process_creation product:linux; detection: selection: Image|endswith: '/bun' ParentImage|contains: '/node'; condition: selection` — deploy via `sigma convert -t splunk` or `sigma convert -t grep` for log-file-based detection. For GitHub Actions secret scoping, use `environment:` declarations in workflow YAML to restrict secret access to specific jobs, preventing a compromised dependency in one job from accessing secrets scoped to another.

Evidence: For lessons-learned documentation and Generation 2 persistence hunting, collect: (1) A full timeline of `npm install` events from all CI pipelines over the past 90 days correlated against the Generation 2 Shai-Hulud 2.0 IOC list from the Microsoft December 2025 advisory — any overlap indicates prior undetected compromise requiring separate incident declaration; (2) Historical GitHub Actions workflow run logs (retained up to 90 days by default) showing any prior execution of the flagged Checkmarx GitHub Action (`ast-github-action`) — this action had CI/CD-level access and any prior runs should be treated as potentially compromised execution environments; (3) Developer workstation `~/.npm/_logs/` install logs and shell history files (`~/.bash_history`, `~/.zsh_history`) for evidence of manual installation of the @bitwarden/cli typosquat outside of automated pipelines; (4) Network flow records or proxy logs showing historical outbound HTTPS to `raw.githubusercontent.com` from build agents at times not correlating to approved workflow runs — this establishes whether the T1102.001 dead-drop channel was active

before this detection event.

Detection Guidance

Primary detection surface is CI/CD pipeline and package manager logs. Look for: (1) npm install or package.json references to @bitwarden/cli, @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service, mbt, ast-github-action, ast-results, or cx-dev-axis, cross-check versions against IOCs from Unit 42 and Wiz Threat Research advisories. (2) Bun runtime process execution (bun, bun.exe) on build agents, developer workstations, or containers not explicitly configured to use Bun; this is a strong detection signal of the bootstrapper. (3) Outbound HTTPS requests to raw.githubusercontent.com or api.github.com from build infrastructure during or after package install phases, TeamPCP uses GitHub commit objects as dead-drop C2 (T1102.001). (4) Post-install script execution in npm, monitor for node scripts spawning shell processes or making network calls during package installation. (5) Cloud provider audit logs: anomalous ListSecrets, GetSecretValue (AWS Secrets Manager), GetSecret (Azure Key Vault), or accessSecretVersion (GCP Secret Manager) calls from build agent service accounts, especially at off-hours or from unfamiliar source IPs. (6) Kubernetes API audit logs for unexpected credential enumeration or service account token reads from build namespaces. SIEM correlation: chain npm install events with subsequent outbound GitHub API calls within the same pipeline execution window as a high-confidence composite indicator.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`raw.githubusercontent.com`	GitHub raw content endpoint used as dead-drop C2 channel for command-and-control communication (T1102.001) — anomalous outbound requests from build agents during or after npm install are high-confidence indicators	HIGH
URL	`https://unit42.paloaltonetworks.com/monitoring-npm-supply-chain-attacks/`	Unit 42 campaign tracking page — consult for current malicious package version hashes and IOC list	HIGH
URL	`https://threats.wiz.io/all-incidents/checkmarx-kics-and-bitwarden-cli-compromised-in-fresh-supply-chain-attack`	Wiz Threat Research incident report covering Checkmarx and Bitwarden CLI compromise — source for specific package version IOCs	HIGH
URL	`https://www.microsoft.com/en-us/security/blog/2025/12/09/shai-hulud-2-0-guidance-for-detecting-investigating-and-defending-against-the-supply-chain-attack/`	Microsoft Generation 2 advisory — contains obfuscation signatures and Bun bootstrapper indicators consistent with Generation 3	HIGH

Framework Mappings

MITRE-ATTACK

- **T1567.001** — Exfiltration to Code Repository
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1140** — Deobfuscate/Decode Files or Information
- **T1539** — Steal Web Session Cookie
- **T1554** — Compromise Host Software Binary
- **T1552.001** — Credentials In Files
- **T1059.007** — JavaScript
- **T1020** — Automated Exfiltration
- **T1102.001** — Dead Drop Resolver
- **T1543** — Create or Modify System Process
- **T1036.005** — Match Legitimate Resource Name or Location
- **T1078.004** — Cloud Accounts
- **T1485** — Data Destruction
- **T1552.004** — Private Keys
- **T1496** — Resource Hijacking
- **T1027** — Obfuscated Files or Information

NIST-800-53R5

- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **IA-5** — Authenticator Management
- **SI-7** — Software, Firmware, and Information Integrity
- **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
T1567.001	Exfiltration to Code Repository	Exfiltration
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1140	Deobfuscate/Decode Files or Information	Defense-Evasion
T1539	Steal Web Session Cookie	Credential-Access
T1554	Compromise Host Software Binary	Persistence
T1552.001	Credentials In Files	Credential-Access
T1059.007	JavaScript	Execution
T1020	Automated Exfiltration	Exfiltration
T1102.001	Dead Drop Resolver	Command-And-Control
T1543	Create or Modify System Process	Persistence
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion
T1078.004	Cloud Accounts	Defense-Evasion
T1485	Data Destruction	Impact
T1552.004	Private Keys	Credential-Access
T1496	Resource Hijacking	Impact
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/2025/12/09/shai-hulud...	T1
	https://unit42.paloaltonetworks.com/npm-supply-chain-attack/	T3
Checkmarx KICS and Bitwarden CLI Compromised in Fresh Supply ...	https://threats.wiz.io/all-incidents/checkmarx-kics-and-bitwarden-c...	T3

Generated 2026-05-02 18:36 UTC by TJS Security Command Center