INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-29 18:48 UTC

SAP CAP npm Supply Chain Attack: Backdoored Packages Steal Cloud Credentials via AI Agent Config Persistence

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0240
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	SAP npm packages: mbt, @cap-js/db-service, @cap-js/postgres, @cap-js/sqlite; GitHub Actions OIDC trusted publishing; VS Code; Claude Code; AWS, Azure, GCP, and Kubernetes credential stores
Published	2026-04-29T12:26:00
Discovery Source	Rss

Executive Summary

On April 29, 2026, a threat actor published backdoored versions of four widely used SAP CAP framework npm packages, exploiting a GitHub Actions misconfiguration to bypass publishing controls and inject credential-harvesting malware. Organizations running SAP CAP-based applications face immediate risk of cloud credential theft across AWS, Azure, GCP, and Kubernetes environments, with over 1,100 exfiltration repositories observed before same-day patching. A persistence mechanism targeting AI coding agent configuration files (VS Code, Claude Code) means malicious instructions may survive package updates, extending the remediation window beyond simple dependency upgrades.

Technical Analysis

On 2026-04-29, threat actor 'miniShai-Hulud', assessed with medium confidence as linked to prior TeamPCP operations [source and methodology to be specified], published malicious versions of four SAP CAP ecosystem npm packages: mbt, @cap-js/db-service, @cap-js/postgres, and @cap-js/sqlite. The initial access vector was a GitHub Actions OIDC trusted publishing misconfiguration (CWE-346): any authorized workflow, not exclusively canonical release workflows, could obtain npm publish tokens, bypassing the intended trust boundary. Payloads were delivered via an obfuscated Bun runtime to evade static analysis. Post-installation execution targeted cloud provider credential stores (AWS, Azure, GCP, Kubernetes) consistent with CWE-522. Post-installation execution triggered CI/CD pipeline injection via modified build configurations (T1195.002, T1554), and

persistence was achieved by writing malicious instructions into VS Code and Claude Code AI agent configuration files (CWE-732, CWE-494), a technique that survives package-level remediation if agent configs are not separately inspected. TLS/certificate validation weaknesses (CWE-295) are assessed to have supported covert exfiltration channel establishment. Over 1,100 exfiltration repositories were identified [per [vendor source], [URL], methodology: [specific behavioral indicators]] before same-day patching. Relevant MITRE techniques include T1195.001/T1195.002 (supply chain compromise), T1552.005/T1552.001/T1552.004 (credential access), T1547/T1543/T1053 (persistence/boot/scheduled task), T1567/T1567.001 (exfiltration), T1059/T1059.001 (execution), T1554 (compromised software dependencies), T1078.004 (valid cloud accounts), T1648 (serverless execution), T1588.003 (obtain capabilities), and T1650 (acquire access). No CVE has been assigned. Affected CWEs: CWE-346, CWE-522, CWE-732, CWE-494, CWE-295. Same-day patching was released; clean package versions must be confirmed against vendor advisories from Snyk, Aikido, StepSecurity, and Semgrep.

Action Checklist

1. Step 1: Containment. Immediately audit all pipeline and developer environments for installed versions of mbt, @cap-js/db-service, @cap-js/postgres, and @cap-js/sqlite published on or around 2026-04-29. Quarantine any build artifacts, containers, or deployed environments that incorporated the affected packages. Rotate all cloud credentials (AWS IAM keys, Azure service principal secrets, GCP service account keys, Kubernetes service account tokens) accessible from affected systems immediately; forensic investigation should proceed in parallel with credential rotation to determine scope of access.
2. Step 2: Detection. Search npm lock files (package-lock.json, yarn.lock, bun.lockb) and SBOM artifacts for the four affected packages with install timestamps on or after 2026-04-29. Query SIEM and cloud provider access logs for anomalous API calls from CI/CD runners, unexpected IAM credential use from build environments, or outbound connections to unknown repositories or hosts from pipeline agents. Check for creation of new GitHub repositories or forks in your org that were not initiated by known team members; over 1,100 exfiltration repos were observed. Review EDR telemetry for Bun runtime execution in unexpected contexts. Behavioral IOCs include: credential file access from npm postinstall scripts, outbound HTTPS to non-standard endpoints from build runners, and modification of VS Code or Claude Code agent configuration files by processes other than the IDE itself.
3. Step 3: Eradication. Upgrade all four affected packages to the clean versions confirmed in vendor advisories (Snyk, Aikido, StepSecurity, Semgrep; validate package signatures and checksums against vendor advisories before deploying). Critically, inspect and sanitize VS Code (.vscode/ settings, extensions, tasks.json) and Claude Code agent configuration files on all developer and CI/CD machines that ran the malicious packages; this incident demonstrates that malicious instructions embedded in these files survive package-level upgrades. Audit and tighten GitHub Actions OIDC trusted publishing configuration: restrict npm publish token issuance to canonical release workflows only, scoped by branch and tag protections (remediate CWE-346). Remove any unrecognized GitHub Actions workflow modifications introduced after 2026-04-28.
4. Step 4: Recovery. After credential rotation, verify that revoked credentials no longer appear in cloud provider access logs. Confirm that CI/CD pipelines rebuild exclusively from verified clean package versions by checking lock file hashes against known-good states. Monitor cloud environments for 30 days post-remediation for signs of persistent access using previously exfiltrated credentials (unexpected API calls, new IAM role assumptions, new service principals). Re-run pipeline builds in isolated environments and validate output artifacts before returning to production.

5. Step 5: Post-Incident. Conduct a GitHub Actions OIDC configuration review across all repositories; implement least-privilege publish token scoping as a pipeline standard. Add AI agent configuration files (VS Code, Claude Code, and equivalents) to your integrity monitoring baseline; this incident demonstrates they are a viable persistence vector. Integrate dependency integrity checks (npm audit signatures, SBOM generation, provenance attestation) into CI/CD gates. If not already in place, implement secrets scanning on repositories and build logs to detect future credential exfiltration attempts. Document this incident as a reference case for supply chain risk in your third-party software intake process.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and cloud provider security teams immediately if CloudTrail, GCP Audit, or Azure AD logs confirm that exfiltrated IAM keys, service principal secrets, or Kubernetes tokens were used to access, exfiltrate, or modify data in production environments — this triggers breach notification assessment under GDPR Article 33, CCPA, and applicable U.S. state laws if PII or regulated data was accessible from the compromised cloud environments.
Recovery Notes	After rotating all cloud credentials, enforce a 30-day enhanced monitoring window across AWS CloudTrail, GCP Cloud Audit Logs, and Azure AD Sign-in logs specifically watching for API calls originating from non-organizational IP ranges using IAM identities associated with CI/CD pipelines — exfiltrated credentials may have been staged for delayed use. Validate that CI/CD rebuild artifacts produce lock file hashes that match known-good pre-April-29 states or confirmed clean post-patch versions per Snyk and Aikido advisories before promoting any artifact to production. Treat any VS Code or Claude Code agent configuration file modified between 2026-04-29 and the date of eradication as potentially compromised and restore from a pre-incident git-tracked baseline.

Forensic Artifacts	npm postinstall execution trace: Node.js process spawn records in OS audit logs (Linux auditd `execve` syscall for node/npm/bun processes, macOS Unified Log process launch events) showing child processes spawned during `npm install` of mbt, @cap-js/db-service, @cap-js/postgres, or @cap-js/sqlite on or after 2026-04-29 — the postinstall scripts are the initial execution point for credential harvesting. | Cloud credential file access events: Sysmon Event ID 11 (FileCreate) or Event ID 10 (ProcessAccess) targeting `%USERPROFILE%\.aws\credentials`, `~/.azure/accessTokens.json`, `~/.config/gcloud/application_default_credentials.json`, and `~/.kube/config` where the accessing process is node.exe, npm.cmd, or bun.exe — these reads would not occur during legitimate npm install operations. | GitHub exfiltration repository creation events: GitHub organization audit log entries for `repo.create` actions performed by OAuth tokens or GitHub Actions bot accounts between 2026-04-29 00:00 UTC and discovery, with repository names not matching organizational naming conventions — the 1,100+ observed exfil repos represent a high-volume artifact unique to this campaign's data staging mechanism. | VS Code and Claude Code agent configuration file modification timestamps: inode ctime and mtime values for `.vscode/settings.json`, `.vscode/tasks.json`, `.vscode/extensions.json`, and `~/.claude/CLAUDE.md` (or `.claude/settings.json`) on developer and CI/CD machines — modification by a process other than `code`, `cursor`, or `claude` with a timestamp correlating to npm install execution is the forensic indicator of the novel AI agent persistence mechanism. | Outbound HTTPS connections from CI/CD runner network segments to GitHub repository clone URLs outside the organization's namespace: network flow logs (VPC Flow Logs for AWS, VPC Flow Logs for GCP, NSG Flow Logs for Azure) showing TCP/443 connections from build runner IPs to github.com resolving to repositories created on or after 2026-04-29 that are not in the organization's known repository list — this network artifact represents the exfiltration channel used to push stolen credentials to the 1,100+ staging repositories.

Per-Action IR Details

Step 1: Containment — Immediately audit all pipeline and developer environments for installed versions of mbt, @cap-js/db-service, @cap-js/postgres, and @cap-js/sqlite published on or around 2026-04-29. Quarantine any build artifacts, containers, or deployed environments that incorporated the affected packages. Rotate all cloud credentials (AWS IAM keys, Azure service principal secrets, GCP service account keys, Kubernetes service account tokens) accessible from affected systems without waiting for forensic confirmation.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST AC-3 (Access Enforcement), CIS 5.3 (Disable Dormant Accounts), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run `npm ls mbt @cap-js/db-service @cap-js/postgres @cap-js/sqlite --all --json 2>/dev/null` in each project root and cross-reference install timestamps with `cat package-lock.json | python3 -m json.tool | grep -A2 '2026-04-29'`. For CI/CD runners without EDR, block outbound HTTPS from pipeline agents to non-approved registries using iptables: `iptables -A OUTPUT -p tcp --dport 443 -m owner --uid-owner ci-runner -j DROP` (allowlist npmjs.com and your artifact proxy only). Use AWS CLI `aws iam list-access-keys --user-name ` and `aws iam delete-access-key` to rotate all IAM keys accessible from affected build environments before forensic completion — credential theft may already be in-progress.

Evidence: Capture BEFORE quarantine: (1) Full `npm ls` dependency tree output from each affected environment saved to file — `npm ls --all --json > npm-tree-$(hostname)-$(date +%s).json`. (2) Node.js postinstall script execution logs — on Linux, check `~/.npm/_logs/` for entries referencing mbt or @cap-js/* install hooks dated 2026-04-29. (3) Bun runtime process invocations — on Linux, `ausearch -c bun --start 04/29/2026 00:00:00` or `journalctl -u bun --since 2026-04-29`; on macOS, check Unified Log: `log show --predicate 'process == "bun"' --start 2026-04-29`. (4) AWS CloudTrail: filter for `GetSecretValue`, `AssumeRole`, `GetCallerIdentity` events from EC2 instance or Lambda function

ARNs associated with CI/CD runners on or after 2026-04-29. (5) Snapshot container images before teardown using `docker commit forensic-preserve-$(date +%s)` to preserve the malicious package state for analysis.

Step 2: Detection — Search npm lock files (package-lock.json, yarn.lock, bun.lockb) and SBOM artifacts for the four affected packages with install timestamps on or after 2026-04-29. Query SIEM and cloud provider access logs for anomalous API calls from CI/CD runners, unexpected IAM credential use from build environments, or outbound connections to unknown repositories or hosts from pipeline agents. Check for creation of new GitHub repositories or forks in your org that were not initiated by known team members — over 1,100 exfiltration repos were observed. Review EDR telemetry for Bun runtime execution in unexpected contexts. Behavioral IOCs include: credential file access from npm postinstall scripts, outbound HTTPS to non-standard endpoints from build runners, and modification of VS Code or Claude Code agent configuration files by processes other than the IDE itself.

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: Deploy Sysmon with SwiftOnSecurity config and add a custom rule targeting FileRead events on `%USERPROFILE%\.aws\credentials`, `%APPDATA%\gcloud\application_default_credentials.json`, and `~/.kube/config` where the originating process image path contains `node`, `npm`, or `bun`. Write a Sigma rule: `title: npm postinstall credential file access | logsource: category: file_access | detection: process_name|contains: ['node','npm','bun'] AND file_path|contains: ['.aws/credentials','application_default_credentials','.kube/config'] | condition: selection`. For GitHub repo enumeration without a SIEM, use the GitHub CLI: `gh api /orgs/{ORG}/repos --paginate --jq '.[].created_at' | grep 2026-04-29` and compare output against your known repo inventory. For lock file scanning across a monorepo, run: `find . -name 'package-lock.json' -exec grep -l 'mbt\|@cap-js' {} \;` then extract resolved versions with `jq '.packages["node_modules/mbt"].version' package-lock.json`.

Evidence: Capture BEFORE remediation: (1) All package-lock.json, yarn.lock, and bun.lockb files from developer workstations and CI/CD runners — hash each with `sha256sum` before any package upgrades alter them. (2) AWS CloudTrail: query for `CreateRepository`, `PutObject`, or `PutItem` events originating from build runner IAM roles between 2026-04-29 00:00 UTC and present, filtering source IP to CI/CD subnet ranges. (3) GCP Cloud Audit Logs: `gcloud logging read 'protoPayload.methodName="storage.objects.create" AND protoPayload.authenticationInfo.principalEmail:"sa-cicd"' --freshness=7d`. (4) VS Code and Claude Code agent config files — collect `~/.vscode/settings.json`, `.vscode/tasks.json`, `.vscode/extensions.json`, and `~/.claude/config` (or equivalent Claude Code config path) with filesystem timestamps preserved using `cp -a` before any sanitization. (5) GitHub Actions audit log: `gh api /orgs/{ORG}/audit-log?phrase=action:workflows.&limit=200` to identify unauthorized workflow modifications introduced after 2026-04-28.

Step 3: Eradication — Upgrade all four affected packages to the clean versions confirmed in vendor advisories (Snyk, Aikido, StepSecurity, Semgrep — URLs listed in source data). Critically, inspect and sanitize VS Code (.vscode/ settings, extensions, tasks.json) and Claude Code agent configuration files on all developer and CI/CD machines that ran the malicious packages — malicious instructions embedded in these files survive package-level upgrades. Audit and tighten GitHub Actions OIDC trusted publishing configuration: restrict npm publish token issuance to canonical release workflows only, scoped by branch and tag protections (remediate CWE-346). Remove any unrecognized GitHub Actions workflow modifications introduced after 2026-04-28.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication and Recovery

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-2 (Baseline Configuration), NIST IR-4 (Incident Handling), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.3 (Address Unauthorized

Software)

Compensating: For AI agent config integrity verification without an enterprise FIM tool, use `sha256sum` baselines: `find ~/.vscode ~/.claude -type f -name '*.json' | xargs sha256sum > vscode-claude-baseline-$(date +%s).txt` then diff against a known-clean developer machine. To detect injected instructions in VS Code tasks.json, grep for common C2 exfiltration patterns: `grep -rE '(curl|wget|Invoke-WebRequest|fetch).*http' ~/.vscode/ .vscode/`. For GitHub Actions OIDC remediation, audit workflow files: `grep -rn 'id-token: write' .github/workflows/` and validate that `permissions.id-token: write` only appears in release workflows explicitly gated by `if: startsWith(github.ref, 'refs/tags/')`. Use `npm pack --dry-run` on upgraded package versions to confirm postinstall script absence before deploying to production.

Evidence: Capture BEFORE eradication: (1) Verbatim content of all VS Code `.vscode/tasks.json`, `.vscode/settings.json`, and Claude Code agent config files (e.g., `~/.claude/CLAUDE.md`, `.claude/settings.json`) with inode change timestamps — run `stat` on each file to record mtime and ctime before modification. (2) GitHub Actions workflow YAML files from `.github/workflows/` directory with git blame output: `git log --all --follow -p .github/workflows/` to identify unauthorized commits post 2026-04-28. (3) npm registry metadata for each affected package version — `npm view mbt@ dist.integrity` to capture the malicious package SHA-512 integrity hash for IOC sharing. (4) The contents of the malicious postinstall scripts: `cat node_modules/mbt/package.json | jq '.scripts.postinstall'` (run on a forensic copy, not production). (5) Kubernetes service account token files at `/var/run/secrets/kubernetes.io/serviceaccount/token` from any pod that ran an affected build, preserved before pod termination.

Step 4: Recovery — After credential rotation, verify that revoked credentials no longer appear in cloud provider access logs. Confirm that CI/CD pipelines rebuild exclusively from verified clean package versions by checking lock file hashes against known-good states. Monitor cloud environments for 30 days post-remediation for signs of persistent access using previously exfiltrated credentials (unexpected API calls, new IAM role assumptions, new service principals). Re-run pipeline builds in isolated environments and validate output artifacts before returning to production.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CP-10 (System Recovery and Reconstitution), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Verify AWS credential rotation effectiveness with: `aws iam list-access-keys --user-name ` confirming only newly issued keys are in `Active` state, then query CloudTrail for any API calls using the old AccessKeyId: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue= | jq '.Events[] | select(.CloudTrailEvent | fromjson | .userIdentity.accessKeyId == "")'`. For GCP, run `gcloud logging read 'protoPayload.authenticationInfo.principalEmail= AND timestamp>=2026-04-29T00:00:00Z' --limit 500` to detect use of rotated service account keys. Validate lock file integrity by comparing SHA-256 hashes of new package-lock.json against a pre-incident baseline stored in git: `git show HEAD~1:package-lock.json | sha256sum` vs `sha256sum package-lock.json`. For isolated pipeline rebuild validation, use `act` (GitHub Actions local runner) to execute workflow in a Docker sandbox: `act push -P ubuntu-latest=ghcr.io/catthehacker/ubuntu:act-latest`.

Evidence: Capture DURING recovery for 30-day monitoring baseline: (1) AWS CloudTrail: configure an EventBridge rule to alert on `AssumeRole` events where the `roleSessionName` matches patterns used by CI/CD runners but the source IP falls outside known runner subnet ranges — this detects use of exfiltrated Kubernetes service account tokens. (2) Azure AD Sign-in logs: filter for service principal authentications from the newly issued client secrets; any authentication against the old secret ID after rotation confirms credential reuse by attacker. (3) GCP Cloud Audit Logs: set a log-based alert on `iam.serviceAccounts.actAs` or `storage.objects.create` from service accounts whose keys were rotated, originating from non-GCP source IPs. (4) Kubernetes API server audit log: monitor for `create` or `bind` operations on ClusterRoleBinding resources that were not initiated by known administrative service accounts — indicates persistence via privilege escalation using exfiltrated tokens. (5) npm lock file git diffs: after pipeline rebuild, `git diff HEAD package-lock.json` to confirm resolved integrity hashes for all four packages match the clean versions per vendor advisories.

Step 5: Post-Incident — Conduct a GitHub Actions OIDC configuration review across all repositories; implement least-privilege publish token scoping as a pipeline standard. Add AI agent configuration files (VS Code, Claude Code, and equivalents) to your integrity monitoring baseline — this attack demonstrated they are a viable persistence vector. Integrate dependency integrity checks (npm audit signatures, SBOM generation, provenance attestation) into CI/CD gates. If not already in place, implement secrets scanning on repositories and build logs to detect future credential exfiltration attempts. Document this incident as a reference case for supply chain risk in your third-party software intake process.

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 RA-3 (Risk Assessment), NIST SA-12 (Supply Chain Protection), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 8.2 (Collect Audit Logs)

Compensating: Implement GitHub Actions OIDC least-privilege by adding explicit permission blocks to all workflow files: enforce `permissions: id-token: write` only in release workflows gated with `if: github.ref_type == 'tag'` and add a required status check using a reusable workflow that validates the calling workflow's ref before minting publish tokens — commit this as a branch protection requirement. For AI agent config FIM without enterprise tooling, add a pre-commit hook: `sha256sum ~/.vscode/settings.json ~/.claude/CLAUDE.md > .git/hooks/vscode-claude-baseline.txt` and a CI step that diffs against baseline on every build. Add SBOM generation to CI/CD using Syft: `syft packages dir:. -o cyclonedx-json > sbom-$(git rev-parse HEAD).json` and validate package provenance attestations with: `npm audit signatures` (requires npm v9+) to detect unsigned or integrity-mismatched packages matching the CWE-494 (Download of Code Without Integrity Check) pattern exploited in this attack. For secrets scanning, deploy `truffleHog filesystem --directory=.` as a pre-push git hook.

Evidence: Document for lessons-learned and future detection baseline: (1) The malicious postinstall script content from each of the four packages — this establishes the behavioral signature (process lineage: npm/node spawning credential file reads + outbound HTTPS) for a future Sysmon or YARA rule targeting similar SAP CAP ecosystem attacks. (2) The specific VS Code and Claude Code config file paths and instruction payloads injected by the malware — document the exact JSON keys or CLAUDE.md directive syntax used for persistence so future FIM baselines explicitly monitor those fields. (3) A list of the observed exfiltration GitHub repository naming patterns from the 1,100+ repos — submit to GitHub Security for takedown and retain as threat intelligence for monitoring your own org's GitHub API audit log. (4) CloudTrail/GCP/Azure log samples showing the anomalous API call patterns generated by the exfiltrated credentials — convert these into detection rules (Sigma or cloud-native alert policies) for long-term monitoring. (5) SBOM snapshot of all four affected package versions with their malicious integrity hashes documented in your vulnerability management system (cross-reference against CVE/GHSA if assigned post-incident) to prevent re-introduction via dependency caching or artifact repositories like Artifactory or Nexus.

Detection Guidance

Primary detection surfaces: (1) Dependency inventory, scan all package-lock.json, yarn.lock, and bun.lockb files for mbt, @cap-js/db-service, @cap-js/postgres, @cap-js/sqlite with resolved timestamps on or after 2026-04-29. (2) CI/CD process telemetry, look for Bun runtime execution spawned from npm postinstall hooks in build logs; query runner process trees for unexpected child processes during package installation. (3) Cloud provider access logs, flag IAM/service account credential use originating from CI/CD runner IP ranges that does not match expected deployment patterns; look for ListBuckets, DescribeInstances, GetSecretValue, or equivalent enumeration calls from build environments. (4) Exfiltration detection, the creation of 1,100+ exfiltration repositories suggests automated infrastructure deployment. High-frequency outbound HTTPS connections from build runners to these repositories should appear as anomalous volume in network flow data. Monitor outbound HTTPS from build runners to GitHub repository endpoints not in your known-good allowlist. (5) AI agent config

integrity, diff current VS Code tasks.json, settings.json, and Claude Code agent config files against version-controlled baselines; look for entries injecting external scripts, modifying trusted hosts, or adding unexpected MCP server endpoints. (6) Credential file access, EDR rules for file reads of ~/.aws/credentials, ~/.azure/, ~/.config/gcloud/, and kubeconfig from processes that are children of npm or bun. Behavioral IOC summary: Bun runtime invoked from npm postinstall context; cloud credential file access by build tooling; outbound connections from build runners to non-allowlisted GitHub repositories; AI agent config modifications by non-IDE processes.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`miniShai-Hulud attributed npm publish account`	Threat actor account used to publish malicious package versions; check npm audit and publish logs for this publisher identity. Exact username should be confirmed against Snyk/Aikido/StepSecurity advisories.	MEDIUM
HASH	`[confirm from Snyk and Semgrep advisories]`	Package integrity hashes for malicious versions of mbt, @cap-js/db-service, @cap-js/postgres, @cap-js/sqlite published 2026-04-29. Hash values must be pulled directly from vendor advisories — not reproduced here to avoid fabrication.	HIGH
URL	`[confirm exfiltration repository patterns from StepSecurity/Aikido advisories]`	Over 1,100 GitHub repositories were used as exfiltration targets. Repository naming patterns and owner accounts should be obtained directly from vendor advisories and used to query GitHub audit logs and network egress logs.	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1648** — Serverless Execution
- **T1588.003** — Code Signing Certificates
- **T1554** — Compromise Host Software Binary
- **T1547** — Boot or Logon Autostart Execution
- **T1078.004** — Cloud Accounts
- **T1195.002** — Compromise Software Supply Chain
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.005** — Cloud Instance Metadata API
- **T1059.001** — PowerShell

- **T1552.004** — Private Keys
- **T1543** — Create or Modify System Process
- **T1053** — Scheduled Task/Job
- **T1567** — Exfiltration Over Web Service
- **T1650** — Acquire Access
- **T1567.001** — Exfiltration to Code Repository
- **T1059** — Command and Scripting Interpreter
- **T1552.001** — Credentials In Files

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **AC-3** — Access Enforcement
- **SC-8** — Transmission Confidentiality and Integrity
- **SC-17** — Public Key Infrastructure Certificates
- **CM-3** — Configuration Change Control
- **AC-6** — Least Privilege
- **IA-5** — Authenticator Management
- **SR-2** — Supply Chain Risk Management Plan
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

- **A02:2021** — Cryptographic Failures
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures
- **A01:2021** — Broken Access Control
- **A04:2021** — Insecure Design

CIS-V8

- **3.10** — Encrypt Sensitive Data in Transit
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **3.3** — Configure Data Access Control Lists
- **5.2** — Use Unique Passwords
- **6.3** — Require MFA for Externally-Exposed Applications
- **7.3** — Perform Automated Operating System Patch Management

- **7.4** — Perform Automated Application Patch Management
- **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

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1648	Serverless Execution	Execution
T1588.003	Code Signing Certificates	Resource-Development
T1554	Compromise Host Software Binary	Persistence
T1547	Boot or Logon Autostart Execution	Persistence
T1078.004	Cloud Accounts	Defense-Evasion
T1195.002	Compromise Software Supply Chain	Initial-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.005	Cloud Instance Metadata API	Credential-Access
T1059.001	PowerShell	Execution
T1552.004	Private Keys	Credential-Access
T1543	Create or Modify System Process	Persistence
T1053	Scheduled Task/Job	Execution

Technique ID	Technique Name	Tactic
T1567	Exfiltration Over Web Service	Exfiltration
T1650	Acquire Access	Resource-Development
T1567.001	Exfiltration to Code Repository	Exfiltration
T1059	Command and Scripting Interpreter	Execution
T1552.001	Credentials In Files	Credential-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/04/sap-npm-packages-compromised-by-m...	T3
Bun-Based Stealer Hits SAP CAP npm Packages - Snyk	https://snyk.io/blog/bun-based-stealer-hits-sap-cap-js-mbt-npm-pack...	T3
Mini Shai-Hulud Targets SAP npm Packages With a Bun-Based ...	https://www.aikido.dev/blog/mini-shai-hulud-has-appeared	T3
Obfuscated Bun Runtime Payloads Hit SAP-Related npm Packages	https://www.stepsecurity.io/blog/a-mini-shai-hulud-has-appeared	T3
SAP Cloud Build Tool Packaged A Mini Shai-Hulud ... - Semgrep	https://semgrep.dev/blog/2026/sap-npm-packages-compromised-in-suppl...	T3

Generated 2026-04-29 18:48 UTC by TJS Security Command Center