INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-30 14:00 UTC

Coordinated npm Supply Chain Campaigns Harvest CI/CD Credentials via Dependency Confusion, Typosquatting, and Compromised Publisher Account

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0384
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	npm ecosystem; Node.js environments; CI/CD pipelines (GitHub Actions, generic); developer workstations (Windows, macOS, Linux); AWS credentials, HashiCorp Vault tokens, GitHub Actions tokens, npm publish tokens; packages spoofing Sberbank SberPay widget and internal enterprise tooling across nine organizational scopes; compromised @antv npm publisher account
Published	2026-05-30T00:06:20+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

Between May 20 and May 29, 2026, Microsoft Threat Intelligence identified three coordinated npm supply chain campaigns delivering malicious packages through dependency confusion, typosquatting, and a compromised publisher account. All three campaigns silently harvest AWS credentials, HashiCorp Vault tokens, GitHub Actions tokens, and npm publish tokens from developer workstations and CI/CD pipelines at install time. Any organization running Node.js build pipelines or using affected packages faces immediate risk of cloud infrastructure takeover and CI/CD pipeline compromise.

Technical Analysis

Microsoft Threat Intelligence documented three structurally related campaigns active May 20-29, 2026. Campaign 1 deployed 33 malicious packages exploiting npm's default resolution behavior (CWE-426: Untrusted Search Path) to override internal private packages via dependency confusion. Campaign 2 deployed 14 typosquatted packages targeting developer install errors. Campaign 3 compromised the legitimate @antv scoped publisher account, injecting malicious payloads into previously trusted packages with existing download history, the highest-trust vector of the three. All campaigns share a common payload delivery mechanism:

postinstall scripts (CWE-506: Embedded Malicious Code) executing at npm install time without user interaction. Payloads perform OS fingerprinting, hostname and username enumeration (T1082, T1033, T1057), then exfiltrate AWS access keys and secrets (T1552.001), HashiCorp Vault tokens, GitHub Actions OIDC tokens (T1552.007), and npm publish tokens over C2 channels (T1041). Obfuscation techniques are present (T1027, T1140). A sub-cluster spoofs Sberbank's SberPay widget targeting Russian-language enterprise scopes. No CVE is assigned; supply chain poisoning campaigns of this class do not receive CVE identifiers. Assessed CVSS base: 9.5 (critical). Relevant CWEs: CWE-312, CWE-426, CWE-494, CWE-506, CWE-829. Primary ATT&CK technique: T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools).

Action Checklist

1. Step 1: Containment. Immediately review all npm dependencies installed or updated between May 20 and May 29, 2026. Cross-reference package names against the 47 malicious packages documented in the Microsoft Threat Intelligence reports (33 dependency confusion + 14 typosquatted). Revoke and rotate any AWS access keys, HashiCorp Vault tokens, GitHub Actions tokens, and npm publish tokens present on affected developer workstations or CI/CD runner environments. Isolate CI/CD pipeline runners that executed npm install during this window pending credential audit. Reference: NIST IR-4 (Incident Handling), CIS 1.1 (Asset Inventory).
2. Step 2: Detection. Review npm install logs and package-lock.json or yarn.lock files for unexpected packages or version changes in the May 20-29 window. Search pipeline logs for postinstall script execution (npm lifecycle events) on packages outside your known-good baseline. Monitor outbound DNS and HTTP requests from build runners for anomalous destinations, particularly short-lived or newly registered domains. Query EDR/endpoint logs for process creation chains originating from node.exe or npm CLI spawning shell commands (cmd.exe, bash, sh, powershell) during install operations, indicative of T1059.007 and T1059.006 activity. Check AWS CloudTrail for access key usage from CI/CD source IPs outside normal baselines, and GitHub audit logs for Actions token usage anomalies. Review @antv scoped packages in your dependency tree for version changes published in this window. Reference: NIST AU-6, AU-12, SI-4; CIS 8.2.
3. Step 3: Eradication. Remove all identified malicious packages from dependency manifests (package.json, package-lock.json, yarn.lock). For @antv packages: pin to versions predating the compromise window or replace with verified clean versions after confirming publisher account recovery. Implement npm package integrity verification using package-lock.json integrity hashes and enforce via CI policy (CWE-494 mitigation). Configure .npmrc to use a private registry mirror (Artifactory, Verdaccio, GitHub Packages) with an allowlist, blocking direct public registry resolution for packages matching internal naming conventions, this directly mitigates CWE-426 and the dependency confusion vector. Reference: NIST CM-7 (Least Functionality), CIS 2.3 (Address Unauthorized Software).
4. Step 4: Recovery. After credential rotation, validate that revoked AWS keys, Vault tokens, and GitHub tokens show no further active sessions in CloudTrail, Vault audit logs, and GitHub audit logs respectively. Re-run CI/CD pipelines from a clean runner image with locked, verified dependencies. Confirm npm publish tokens have been regenerated and old tokens are fully revoked in the npm registry account settings. Validate that private package registry mirror is resolving all internal packages correctly before restoring production pipeline operations. Reference: NIST IR-4, SI-7 (Software, Firmware, and Information Integrity); CIS 7.1.

5. Step 5: Post-Incident. Conduct a formal review of dependency management controls. Implement mandatory private registry proxying to eliminate public registry dependency confusion exposure (mitigates T1195.001 at infrastructure level). Enforce subresource integrity or lockfile-based hash verification for all npm installs in CI/CD (addresses CWE-494). Apply least-privilege principles to CI/CD pipeline credentials, use short-lived OIDC tokens scoped to minimum permissions rather than long-lived static keys (NIST AC-6, CIS 5.4, D3-CRO: Credential Rotation, D3-CH: Credential Hardening). Enable npm publisher account MFA organization-wide and audit all third-party scoped publisher dependencies for account security posture (CIS 6.3, CIS 6.5, D3-MFA). Evaluate dependency confusion risk across all internal package namespaces and register shadow public packages where feasible to block future confusion attacks.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and breach notification counsel immediately if AWS CloudTrail, Vault audit logs, or GitHub audit logs confirm that any harvested credential was actively used by the attacker post-exfiltration, as this elevates the incident from credential exposure to confirmed unauthorized access with potential regulatory notification obligations under GDPR, CCPA, or applicable state breach notification laws depending on data accessible via the compromised credentials.
Recovery Notes	After credential rotation and pipeline restoration, maintain enhanced monitoring of all newly issued AWS IAM keys, Vault tokens, and GitHub Actions tokens for a minimum of 30 days, specifically watching for usage patterns that match the original CI/CD source IPs or times — attackers who harvested credentials sometimes delay their use to evade temporal correlation. Continuously monitor npmjs.com for re-publication of the 47 malicious package names (attackers frequently re-upload under the same names after takedown) using automated npm registry watch scripts or Socket.dev free tier. Verify that all internal package namespaces confirmed as vulnerable to dependency confusion during the post-incident review have either been shadow-registered on the public npm registry or are fully blocked by private registry allowlist policy before any pipeline is returned to unrestricted operation.

Forensic Artifacts	node_modules/.package-lock.json and package-lock.json on each affected runner and developer workstation — contains exact resolved package versions, registry sources, and sha512 integrity hashes for every installed package in the May 20–29 window, enabling definitive confirmation of which of the 47 malicious packages were installed | npm verbose install logs at ~/.npm/_logs/*.log (Linux/macOS) or %AppData%\npm-cache_logs*.log (Windows) — these logs record the exact postinstall script command lines executed by malicious packages, including the credential harvesting commands targeting AWS_ACCESS_KEY_ID, VAULT_TOKEN, GITHUB_TOKEN, and NPM_TOKEN environment variables | AWS CloudTrail logs filtered for GetCallerIdentity, AssumeRole, ListBuckets, and CreateSession events originating from CI/CD runner source IPs during and after the May 20–29 window — confirms whether harvested AWS credentials were exercised by the attacker and identifies what AWS resources were accessed | Sysmon Event ID 1 (Process Create) or auditd execve records showing node.exe or npm spawning cmd.exe, powershell.exe, bash, or sh as child processes during npm install operations — directly evidences T1059.007 (JavaScript) and T1059.006 (Python) postinstall execution chains used by the malicious packages to execute credential harvesting scripts | DNS query logs and outbound HTTP/HTTPS connection logs from build runner hosts for the compromise window — the malicious postinstall scripts exfiltrate harvested credentials via HTTP POST or DNS exfiltration to attacker-controlled domains, making these logs the primary evidence of successful credential exfiltration and the sole source for attacker C2 infrastructure identification

Per-Action IR Details

Step 1: Containment — Immediately audit all npm dependencies installed or updated between May 20 and May 29, 2026. Cross-reference package names against the 47 malicious packages documented in the Microsoft Threat Intelligence reports (33 dependency confusion + 14 typosquatted). Revoke and rotate any AWS access keys, HashiCorp Vault tokens, GitHub Actions tokens, and npm publish tokens present on affected developer workstations or CI/CD runner environments. Isolate CI/CD pipeline runners that executed npm install during this window pending credential audit. Reference: NIST IR-4 (Incident Handling), CIS 1.1 (Asset Inventory).

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy: isolate affected CI/CD runners and revoke harvested credentials before attacker leverages exfiltrated AWS keys, Vault tokens, or GitHub Actions tokens for lateral movement or supply chain persistence

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management) — disable or invalidate compromised AWS IAM keys and npm publish tokens, NIST AC-17 (Remote Access) — isolate CI/CD runners from production networks pending credential audit, CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory) — enumerate all runner environments that executed npm install in the May 20–29 window, CIS 6.2 (Establish an Access Revoking Process) — revoke all credentials present in runner environments confirmed to have installed any of the 47 malicious packages

Compensating: Run 'npm ls --all --json > installed_packages.json' on each developer workstation and CI runner, then diff against the Microsoft-published IOC list using a Python script (json + csv comparison). For AWS key revocation without a SIEM: use 'aws iam list-access-keys --user-name ' and 'aws iam update-access-key --status Inactive' for each affected identity. For GitHub Actions tokens, navigate to Settings > Developer Settings > Personal Access Tokens and revoke all tokens scoped to the affected repositories. Use osquery on runners: 'SELECT name, version, path FROM npm_packages WHERE install_time BETWEEN AND ;' to enumerate installed packages without an EDR.

Evidence: BEFORE isolating runners, capture: (1) full contents of node_modules/.package-lock.json and package-lock.json on each runner — these record exact resolved versions and integrity hashes; (2) npm install log at ~/.npm/_logs/*.log or pipeline stdout — contains timestamps and package names of every installed package; (3) /tmp or %TEMP% directory listing for transient exfiltration script artifacts dropped by postinstall hooks (look for files with .js or no extension created in the May 20–29 window); (4) AWS CloudTrail GetCallerIdentity and AssumeRole events from

CI/CD source IPs for the compromise window — these show whether harvested keys were already used; (5) environment variable snapshots — the malicious postinstall scripts specifically targeted AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, VAULT_TOKEN, GITHUB_TOKEN, and NPM_TOKEN environment variables, so capture 'printenv' output or CI runner environment exports as evidence of what credentials were exposed.

Step 2: Detection — Audit npm install logs and package-lock.json or yarn.lock files for unexpected packages or version changes in the May 20–29 window. Search pipeline logs for postinstall script execution (npm lifecycle events) on packages outside your known-good baseline. Monitor outbound DNS and HTTP requests from build runners for anomalous destinations, particularly short-lived or newly registered domains. Query EDR/endpoint logs for process creation chains originating from node.exe or npm CLI spawning shell commands (cmd.exe, bash, sh, powershell) during install operations — indicative of T1059.007 and T1059.006 activity. Check AWS CloudTrail for access key usage from CI/CD source IPs outside normal baselines, and GitHub audit logs for Actions token usage anomalies. Review @antv scoped packages in your dependency tree for version changes published in this window. Reference: NIST AU-6, AU-12, SI-4; CIS 8.2.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: correlate npm lifecycle event logs, process creation telemetry, and credential usage logs to determine which of the three campaign vectors (dependency confusion, typosquatting, compromised @antv publisher) triggered and whether exfiltration occurred

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting) — review npm install logs, CloudTrail, and GitHub audit logs for the May 20–29 window, NIST AU-12 (Audit Record Generation) — verify that pipeline logging captured postinstall script execution and outbound network activity during npm install operations, NIST SI-4 (System Monitoring) — monitor for node.exe or npm spawning cmd.exe, bash, sh, or powershell as child processes during install, and outbound HTTP/DNS from build runners to newly registered domains, CIS 8.2 (Collect Audit Logs) — ensure CI/CD runner audit logs were enabled and retained for the full compromise window

Compensating: Without EDR, deploy Sysmon on Windows runners using the SwiftOnSecurity config (sysmonconfig-export.xml) and filter Event ID 1 (Process Create) for node.exe or npm.cmd as parent process with cmd.exe, powershell.exe, or wscript.exe as child — this directly detects the T1059.007/T1059.006 postinstall execution chain. On Linux runners, use auditd with '-a always,exit -F exe=/usr/bin/node -S execve' to capture child process spawning. For network detection without a SIEM, run Wireshark or tcpdump on the runner's interface during a controlled re-execution of the suspect install in an isolated environment, filtering for DNS queries and HTTP POST to non-npm-registry destinations. For @antv package version drift, run: 'npm view @antv/ time --json' to retrieve the publish history and confirm whether a new version was published between May 20–29. Use the free Sigma rule 'proc_creation_win_node_susp_child_process' (SigmaHQ community rules) converted to Windows Event Log format for offline log scanning with hayabusa.

Evidence: BEFORE completing detection scope, preserve: (1) ~/.npm/_logs/*.log files from all developer workstations — npm verbose logs record exact postinstall script command lines executed by each package; (2) GitHub Actions workflow run logs (Settings > Actions > Workflow runs) for any workflow that called 'npm install' or 'npm ci' between May 20–29 — download the full log archive before GitHub auto-purges; (3) DNS query logs from build runner hosts or network perimeter — the malicious packages' postinstall scripts beacon to attacker-controlled C2 domains to exfiltrate harvested credentials, so DNS logs will contain the exfiltration destination even if HTTP is encrypted; (4) Sysmon Event ID 3 (Network Connection) or Windows Firewall logs for outbound connections from node.exe or cmd.exe spawned by npm during the window; (5) AWS CloudTrail 'CreateSession', 'GetCallerIdentity', and 'ListBuckets' events for any IAM identity associated with keys present in runner environments — these confirm whether harvested AWS credentials were exercised by the attacker.

Step 3: Eradication — Remove all identified malicious packages from dependency manifests (package.json, package-lock.json, yarn.lock). For @antv packages: pin to versions predating the compromise window or replace with verified clean versions after confirming publisher account recovery. Implement npm package integrity verification using package-lock.json integrity hashes and enforce via CI policy (CWE-494 mitigation). Configure .npmrc to use a private registry mirror (Artifactory, Verdaccio, GitHub Packages) with an allowlist, blocking direct public registry resolution for packages matching internal naming conventions — this directly

mitigates CWE-426 and the dependency confusion vector. Reference: NIST CM-7 (Least Functionality), CIS 2.3 (Address Unauthorized Software).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication: remove all 47 malicious packages from dependency trees and implement registry controls that prevent re-introduction of dependency confusion and typosquatted packages that were the delivery mechanism for this campaign

Controls: NIST CM-7 (Least Functionality) — configure .npmrc to restrict package resolution to a private registry mirror, blocking direct npmjs.com resolution for internal-namespace packages, NIST SI-2 (Flaw Remediation) — replace compromised @antv package versions with publisher-confirmed clean versions and update lockfiles with verified integrity hashes, NIST CM-3 (Configuration Change Control) — enforce lockfile-based integrity verification in CI/CD as a policy gate before pipeline execution proceeds, CIS 2.3 (Address Unauthorized Software) — remove all 47 malicious packages identified by Microsoft Threat Intelligence from all dependency manifests and node_modules directories, CIS 2.2 (Ensure Authorized Software is Currently Supported) — validate that retained @antv packages are from the confirmed-clean publisher account post-recovery

Compensating: For teams without Artifactory: deploy Verdaccio (free, open-source) as a local npm proxy with a scoped allowlist — configure .npmrc with 'registry=http://localhost:4873' and set Verdaccio's uplink to only proxy known-safe scopes. To enforce lockfile integrity without a commercial CI gate, add a pre-install npm script: 'node -e "require('crypto'); const lock = require('./package-lock.json'); /* hash validation logic */"' or use the npm-audit-ci tool (free) with a custom blocklist containing the 47 malicious package names. To verify @antv package authenticity before pinning, run 'npm view @antv/ dist.integrity' and compare the sha512 hash against the known-good version from before May 20, 2026. For dependency confusion namespace squatting, run 'npm search ' to verify no public package exists for your internal naming conventions, and if absent, publish empty placeholder packages to npm under your internal names to block confusion attacks.

Evidence: BEFORE removing packages, collect: (1) hash of every malicious package tarball in node_modules/.cache/npm — these are the exact malicious artifacts and should be preserved for forensic comparison against Microsoft's IOC hashes; (2) full content of any .js files dropped in /tmp, %APPDATA%, or the project root by postinstall hooks — the exfiltration scripts themselves are forensic evidence of attacker tooling and C2 infrastructure; (3) a copy of the compromised package's package.json 'scripts.postinstall' field — this contains the exact credential harvesting command and should be documented before removal; (4) node_modules//index.js or the entrypoint file — preserves the malicious payload for YARA rule development and future detection; (5) environment variable state at the time of install (reconstructed from CI/CD runner logs) confirming which of AWS_ACCESS_KEY_ID, VAULT_TOKEN, GITHUB_TOKEN, NPM_TOKEN were populated and therefore harvestable.

Step 4: Recovery — After credential rotation, validate that revoked AWS keys, Vault tokens, and GitHub tokens show no further active sessions in CloudTrail, Vault audit logs, and GitHub audit logs respectively. Re-run CI/CD pipelines from a clean runner image with locked, verified dependencies. Confirm npm publish tokens have been regenerated and old tokens are fully revoked in the npm registry account settings. Validate that private package registry mirror is resolving all internal packages correctly before restoring production pipeline operations. Reference: NIST IR-4, SI-7 (Software, Firmware, and Information Integrity); CIS 7.1.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery: restore CI/CD pipeline integrity from a clean runner image with verified lockfiles after confirming all harvested credentials (AWS keys, Vault tokens, GitHub Actions tokens, npm publish tokens) show no continued attacker activity in their respective audit logs

Controls: NIST IR-4 (Incident Handling) — execute recovery steps per the IR plan, validating each credential type in its authoritative audit log before restoring pipeline operations, NIST SI-7 (Software, Firmware, and Information Integrity) — verify clean runner image integrity and lockfile hash consistency before re-enabling production CI/CD runs, NIST AC-2 (Account Management) — confirm all revoked AWS IAM keys, npm tokens, and GitHub PATs are in 'Inactive' or 'Deleted' state with no new sessions appearing post-revocation, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — validate the private registry mirror is correctly resolving all dependency names before restoring production pipelines, CIS 7.2 (Establish and Maintain a Remediation Process) — document credential rotation and pipeline restoration steps with timestamps for post-incident evidence

Compensating: Without a commercial vault or secrets manager, validate AWS key revocation by running 'aws iam get-access-key-last-used --access-key-id ' — the 'LastUsedDate' should not update after the revocation timestamp; any post-revocation activity indicates attacker persistence with a cloned credential. For HashiCorp Vault token validation, use 'vault token lookup ' — revoked tokens return '403 permission denied'; also check Vault audit log at /var/log/vault/vault_audit.log for any auth attempts using the old token after revocation. For GitHub Actions, use the GitHub REST API: 'GET /orgs/{org}/audit-log?phrase=token' to retrieve token usage events and verify no activity after revocation. To build a clean runner image without a paid CI platform, use a Docker container with a pinned base image (e.g., node:20.x-alpine@sha256:) and 'npm ci --ignore-scripts' to install from the verified lockfile while blocking all postinstall execution.

Evidence: During recovery validation, collect and retain: (1) AWS CloudTrail 'UpdateAccessKey' event (status: Inactive) with timestamp as proof of revocation, and a 24-hour window of GetCallerIdentity events post-revocation confirming the old key is not being used; (2) HashiCorp Vault audit log entries showing 'token revoked' events and any subsequent failed auth attempts using old tokens — failed attempts post-revocation indicate attacker is retrying harvested credentials; (3) GitHub audit log export (Settings > Audit Log > Export) filtered for the compromised token IDs showing revocation event and confirming no post-revocation usage; (4) npm registry account activity log showing old NPM_TOKEN revocation and new token generation timestamps — accessible via npmjs.com account settings audit view; (5) SHA-256 hash of the clean runner Docker image or VM snapshot used for the first post-recovery pipeline run, establishing a verified clean baseline.

Step 5: Post-Incident — Conduct a formal review of dependency management controls. Implement mandatory private registry proxying to eliminate public registry dependency confusion exposure (mitigates T1195.001 at infrastructure level). Enforce subresource integrity or lockfile-based hash verification for all npm installs in CI/CD (addresses CWE-494). Apply least-privilege principles to CI/CD pipeline credentials — use short-lived OIDC tokens scoped to minimum permissions rather than long-lived static keys (NIST AC-6, CIS 5.4, D3-CRO: Credential Rotation, D3-CH: Credential Hardening). Enable npm publisher account MFA organization-wide and audit all third-party scoped publisher dependencies for account security posture (CIS 6.3, CIS 6.5, D3-MFA). Evaluate dependency confusion risk across all internal package namespaces and register shadow public packages where feasible to block future confusion attacks.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity: formalize lessons learned from this three-vector npm supply chain campaign and implement structural controls that eliminate the dependency confusion vector (CWE-426), block typosquatting via registry allowlisting, and harden publisher account security to prevent recurrence of the @antv-style compromised publisher attack

Controls: NIST AC-6 (Least Privilege) — replace long-lived static AWS IAM keys and GitHub PATs in CI/CD with short-lived OIDC tokens scoped to minimum required permissions per pipeline, NIST CM-7 (Least Functionality) — mandate private registry proxying via .npmrc configuration as an organizational standard, blocking direct npmjs.com resolution for all internal-namespace packages, NIST SI-7 (Software, Firmware, and Information Integrity) — enforce lockfile integrity hash verification as a required CI/CD gate, rejecting any install where package hashes do not match package-lock.json, NIST RA-3 (Risk Assessment) — evaluate all internal package namespaces for dependency confusion exposure and document residual risk for namespaces where shadow registration is not feasible, CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts) — restrict npm publish token scope to specific packages only; do not use organization-wide publish tokens in CI/CD, CIS 6.3 (Require MFA for Externally-Exposed Applications) — enforce MFA on all npmjs.com publisher accounts, especially those controlling scoped packages used across the organization, CIS 6.5 (Require MFA for Administrative Access) — require MFA for all npm organization owner and team admin accounts to prevent recurrence of the @antv compromised publisher account vector, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — incorporate npm publisher account security audits and dependency confusion namespace sweeps into the quarterly vulnerability management cycle

Compensating: To implement OIDC-based short-lived tokens in GitHub Actions without cost: configure the workflow with 'permissions: id-token: write' and use the aws-actions/configure-aws-credentials action with role-to-assume — this eliminates static AWS_ACCESS_KEY_ID/AWS_SECRET_ACCESS_KEY from runner environments entirely, removing the primary credential harvesting target for this campaign's postinstall scripts. For npm publisher MFA without an enterprise SSO platform: enable 2FA at npmjs.com account settings (two-factor authentication > Auth and Writes)

for every account with publish rights — this is free and directly prevents the @antv compromised publisher account vector. To audit all internal package namespaces for confusion risk: run 'npm search --json' or query the npm registry API ('https://registry.npmjs.org/-/v1/search?text=') to identify whether public packages exist for your internal naming conventions. Use the free retire.js CLI tool integrated into CI to flag packages with known-malicious versions going forward.

Evidence: For the post-incident lessons learned record, collect and archive: (1) the complete Microsoft Threat Intelligence IOC list (all 47 malicious package names and versions) as a permanent reference artifact for future dependency audits; (2) a point-in-time export of all organization CI/CD environment variable configurations showing which secrets were exposed in runner environments — this documents the blast radius and informs least-privilege redesign; (3) Vault and AWS IAM policy exports showing permission scope of all rotated credentials — required to assess what data and infrastructure was accessible with the harvested credentials had the attacker exercised them; (4) timeline reconstruction of the first malicious package install to credential rotation completion — this measures detection gap and informs SLA targets for future supply chain incidents; (5) npm organization audit log export showing all publish events and account access during the May 20–29 window — establishes whether any internally-controlled publisher accounts were targeted alongside the confirmed @antv compromise.

Detection Guidance

Primary detection surface is CI/CD pipeline execution logs and npm lifecycle event telemetry. Key behavioral indicators: (1) postinstall script execution on packages not in your approved baseline, search pipeline logs for 'postinstall' lifecycle events on unfamiliar package names; (2) node.exe or npm CLI spawning child processes (sh, bash, cmd.exe, powershell) outside of your known build scripts, correlates to T1059.007/T1059.006; (3) outbound HTTP/HTTPS or DNS requests from build runner processes to domains not in your CI/CD allow-list during or immediately after npm install operations, correlates to T1041 exfiltration. For AWS exposure: query CloudTrail for GetCallerIdentity, AssumeRole, or ListBuckets API calls originating from IP ranges associated with CI/CD runners but outside expected timeframes or using access key IDs not provisioned for that pipeline. For GitHub Actions: review audit log for token usage events from unexpected source IPs or repositories. For Vault: review audit device logs for token lookups from build runner hostnames not previously observed. IOC patterns to hunt: package names closely resembling your internal private package names (dependency confusion indicator); package names with character substitutions on popular packages (typosquatting indicator, e.g., single character swaps, added/removed hyphens); unexpected version bumps on @antv scoped packages published May 20-29, 2026. Reference: NIST AU-6, AU-12, SI-4; CIS 8.2; D3-LAM (Local Account Monitoring); D3-SFA (System File Analysis) for lockfile tampering detection.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.microsoft.com/en-us/security/blog/2026/05/29/33-malicious-npm-packages-abuse-dependency-confusion-profile-developer-environments/`	Microsoft Threat Intelligence report: 33 malicious dependency confusion npm packages, developer environment profiling and credential exfiltration — package name list documented here	HIGH

Type	Value	Context	Confidence
URL	`https://www.microsoft.com/en-us/security/blog/2026/05/28/typosquatted-npm-packages-used-steal-cloud-ci-cd-secrets/`	Microsoft Threat Intelligence report: 14 typosquatted npm packages targeting cloud and CI/CD credential theft — package name list documented here	HIGH
URL	`https://www.microsoft.com/en-us/security/blog/2026/05/20/mini-shai-hulud-compromised-antv-npm-packages-enable-ci-cd-credential-theft/`	Microsoft Threat Intelligence report: compromised @antv publisher account enabling malicious payload injection into trusted packages — affected package versions documented here	HIGH
URL	`https://safedep.io/ti/packages/npm/@sber-ecom-core/sberpay-widget`	SafeDep threat intelligence entry for @sber-ecom-core/sberpay-widget — malicious package spoofing Sberbank SberPay widget; label: search-retrieved, recommend human validation	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1566** — Phishing
- **T1140** — Deobfuscate/Decode Files or Information
- **T1036.003** — Rename Legitimate Utilities
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.001** — Credentials In Files
- **T1041** — Exfiltration Over C2 Channel
- **T1078** — Valid Accounts
- **T1057** — Process Discovery
- **T1078.004** — Cloud Accounts
- **T1083** — File and Directory Discovery
- **T1059.006** — Python
- **T1552.004** — Private Keys
- **T1082** — System Information Discovery
- **T1059.007** — JavaScript
- **T1552.007** — Container API
- **T1548.003** — Sudo and Sudo Caching
- **T1033** — System Owner/User Discovery
- **T1027** — Obfuscated Files or Information

NIST-800-53R5

- **AT-2** — Literacy Training and Awareness

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-8** — Spam Protection
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **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

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **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

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

ISO-27001-2022

- **A.5.34** — Privacy and protection of personal information
- **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
T1566	Phishing	Initial-Access

Technique ID	Technique Name	Tactic
T1140	Deobfuscate/Decode Files or Information	Defense-Evasion
T1036.003	Rename Legitimate Utilities	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1041	Exfiltration Over C2 Channel	Exfiltration
T1078	Valid Accounts	Defense-Evasion
T1057	Process Discovery	Discovery
T1078.004	Cloud Accounts	Defense-Evasion
T1083	File and Directory Discovery	Discovery
T1059.006	Python	Execution
T1552.004	Private Keys	Credential-Access
T1082	System Information Discovery	Discovery
T1059.007	JavaScript	Execution
T1552.007	Container API	Credential-Access
T1548.003	Sudo and Sudo Caching	Privilege-Escalation
T1033	System Owner/User Discovery	Discovery
T1027	Obfuscated Files or Information	Defense-Evasion

Sources

Source	URL	Tier
Microsoft Security Blog	https://www.microsoft.com/en-us/security/blog/2026/05/29/33-malicio...	T1
	https://www.microsoft.com/en-us/security/blog/2026/05/29/33-malicio...	T1
	https://www.microsoft.com/en-us/security/blog/2026/05/28/typosquatt...	T1
	https://www.microsoft.com/en-us/security/blog/2026/05/20/mini-shai-...	T1
Is @sber-ecom-core/sberpay-widget malicious? — npm - SafeDep	https://safedep.io/ti/packages/npm/@sber-ecom-core/sberpay-widget	T3

DISCLAIMER

This intelligence report is produced by Tech Jacks Solutions Security Command Center (SCC) for informational purposes only. It does not constitute professional security advice, legal counsel, or an incident response engagement. The information herein is derived from publicly available sources and AI-assisted analysis; while every effort is made to ensure accuracy, Tech Jacks Solutions makes no warranties regarding completeness or timeliness. Organizations should conduct their own validation and consult qualified security professionals before taking action based on this report. Tech Jacks Solutions is not liable for any damages resulting from the use of this information.

Generated 2026-05-30 14:00 UTC by TJS Security Command Center