INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-30 06:22 UTC

Dual npm Supply Chain Campaigns Target Developer Environments with Reconnaissance Payloads and Cloud Credential Theft

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0383
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	npm registry, Node.js developer environments, CI/CD pipelines, enterprise build systems; impersonated scopes include Sberbank (SberPay widget) and unnamed enterprise organizations
Published	2026-05-30T00:06:20+00:00
Discovery Source	Rss:T1 Threatintel

Executive Summary

Microsoft Threat Intelligence identified two coordinated malicious npm package campaigns published May 28-29, 2026, targeting enterprise software development pipelines. The first campaign used 33 dependency-confusion packages to silently profile developer environments for staged follow-on attacks; the second deployed 14 typosquatting packages to steal AWS credentials, HashiCorp Vault tokens, and CI/CD secrets at install time. Any organization with Node.js developers, automated build systems, or cloud-connected CI/CD pipelines faces direct risk of credential theft and potential full cloud environment compromise.

Technical Analysis

Two concurrent npm supply chain campaigns were identified by Microsoft Threat Intelligence on May 28-29, 2026. Campaign 1 comprised 33 packages exploiting dependency confusion (CWE-426) against internal package naming conventions, executing silently via npm lifecycle hooks (preinstall/postinstall) to harvest system metadata, OS, hostname, username, environment variables, for follow-on targeting (T1082, T1119, T1195.001, T1059.007). Campaign 2 comprised 14 typosquatting packages (CWE-829, T1036.001) that exfiltrated AWS credentials, HashiCorp Vault tokens, and CI/CD secrets at install time via obfuscated JavaScript payloads (T1027) communicating over HTTP/S (T1071.001), abusing cloud instance metadata endpoints (T1552.005, T1552.007, CWE-200: Exposure of Sensitive Information) and credential files (T1552.001, T1528). Both

campaigns abuse lifecycle hook execution (T1543, T1053) and shared evasion techniques, suggesting coordinated or parallel threat actor activity. Impersonated scopes include Sberbank's SberPay widget. Known threat actor npm aliases: mr.4nd3r50n, ce-rwb, t-in-one, vpmdhaj. Packages have been removed from the registry, but exfiltrated credentials and profiled environment data remain active risk. No CVE assigned; CWE classifications include CWE-494 (Download of Code Without Integrity Check), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-426 (Untrusted Search Path), CWE-200 (Exposure of Sensitive Information).

Action Checklist

1. Step 1: Containment, Audit all npm install logs from May 25-31, 2026 across developer workstations, build servers, and CI/CD runners. Identify any packages published by aliases mr.4nd3r50n, ce-rwb, t-in-one, or vpmdhaj, or any packages matching known impersonated scopes (SberPay-related naming). Immediately revoke and rotate any AWS IAM credentials, HashiCorp Vault tokens, and CI/CD secrets (GitHub Actions, Jenkins, GitLab CI) accessible from affected systems. Apply NIST IR-4 (Incident Handling) procedures to scope the blast radius before allowing affected pipelines to resume.
2. Step 2: Detection, Query SIEM and endpoint logs for npm preinstall/postinstall script execution events spawning child processes (node, sh, bash, cmd) or making outbound HTTP/S connections from build agents. Search package-lock.json and npm audit logs for the 47 known malicious package names (obtain full IOC list from Microsoft Security Blog source). Enable NIST AU-2 (Event Logging) and CIS 8.2 (Collect Audit Logs) on all CI/CD nodes if not already active. Check AWS CloudTrail for GetCallerIdentity, AssumeRole, and ListBuckets calls originating from unexpected EC2 instance metadata service (IMDS) queries or developer workstation IPs. Alert on environment variable enumeration patterns consistent with T1082 and T1119.
3. Step 3: Eradication, Remove all identified malicious packages from node_modules directories, package.json, and package-lock.json across affected systems. Rotate all credentials confirmed or suspected to have been accessible during any malicious install event: AWS access keys (IAM), HashiCorp Vault tokens, CI/CD pipeline secrets, and any .npmrc tokens. Enforce npm package integrity verification using lockfile integrity hashes (npm ci over npm install) per CWE-494 remediation guidance. Apply CIS 4.6 (Securely Manage Enterprise Assets and Software) by locking internal package scope resolution to a private registry (Artifactory, Nexus, or AWS CodeArtifact) with explicit scope mapping to prevent dependency confusion. Disable or restrict npm lifecycle script execution in CI/CD environments using --ignore-scripts flag where feasible.
4. Step 4: Recovery, Validate that all rotated credentials are functional and that no residual access remains on compromised IAM roles or Vault paths. Re-run builds from clean environments with verified package manifests. Monitor AWS CloudTrail, Vault audit logs, and CI/CD pipeline logs for 30 days post-remediation for anomalous access patterns consistent with delayed exploitation using profiled environment data from Campaign 1. Confirm private registry scope resolution is enforced and test dependency confusion controls using a benign internal package name against the public registry. Apply NIST SI-4 (System Monitoring) to maintain elevated detection posture on build infrastructure.
5. Step 5: Post-Incident, Conduct a supply chain security review against NIST SP 800-161r1 (Cyber Supply Chain Risk Management). Implement software composition analysis (SCA) tooling in CI/CD pipelines to detect malicious or untrusted packages before install. Enforce CIS 2.1 (Establish and Maintain a Software Inventory) and CIS 2.3 (Address Unauthorized Software) for npm dependencies. Establish a private npm registry with allowlist-based scope controls (AC-3, AC-4) to eliminate public registry

dependency confusion risk. Review and restrict environment variable exposure in CI/CD pipeline configurations to limit credential harvest surface (AC-6, Least Privilege). Document lessons learned and update software supply chain risk in the organizational risk register.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal/compliance immediately if AWS CloudTrail confirms GetCallerIdentity or AssumeRole calls using keys accessible during the malicious npm install window, if any exfiltrated CI/CD secrets include production environment credentials, or if build artifacts from the May 25-31 window were deployed to production systems (triggering potential breach notification obligations under applicable data protection regulations).
Recovery Notes	Re-enable affected CI/CD pipelines only after confirming private registry scope enforcement is validated via dependency confusion test, all credential rotations are complete across AWS IAM, HashiCorp Vault, and CI/CD secret stores, and clean `npm ci` rebuilds succeed from verified lockfiles. Maintain elevated monitoring on AWS CloudTrail for GetCallerIdentity, AssumeRole, and S3 data access events for a minimum of 30 days post-remediation, as Campaign 1's reconnaissance payload was designed to profile environments for delayed, targeted follow-on attacks using collected data. Treat any anomalous access to AWS resources or Vault paths from previously affected developer IPs during the 30-day window as a confirmed secondary incident requiring full IR restart.
Forensic Artifacts	npm debug logs at ~/.npm/_logs/*.log on developer workstations and CI/CD runners: these capture exact package names, resolved registry URLs, install timestamps, and any lifecycle script stdout/stderr output from malicious preinstall/postinstall hooks executed by Campaign 1 and Campaign 2 packages during the May 25-31 window | AWS CloudTrail GetCallerIdentity and AssumeRole events filtered to May 25-31: Campaign 2 specifically used GetCallerIdentity as a live-validation step to confirm stolen AWS_ACCESS_KEY_ID/AWS_SECRET_ACCESS_KEY values before exfiltrating them, making this API call a high-fidelity forensic fingerprint of successful credential theft | Process execution records (Sysmon Event ID 1 on Windows or auditd execve on Linux) showing node.exe or node as parent process spawning curl, wget, sh, bash, or PowerShell during npm install operations: these represent the malicious lifecycle hook execution chain used by both campaigns to run reconnaissance (Campaign 1) and credential exfiltration (Campaign 2) payloads | Temporary file artifacts written by Campaign 1 reconnaissance payloads: search /tmp/, $TMPDIR, and %TEMP% for JSON, base64, or plain-text files created by node processes between May 25-31 containing hostname, username, environment variable enumerations, or directory listings consistent with T1082 (System Information Discovery) and T1119 (Automated Collection) | package-lock.json files from all projects with npm installs during May 25-31: the resolved registry URL field in each dependency entry will show whether packages resolved to registry.npmjs.org (public — dependency confusion exploitation) instead of your private registry, providing definitive proof of the attack vector and the exact package versions installed from malicious sources

Per-Action IR Details

Step 1: Containment — Audit all npm install logs from May 25-31, 2026 across developer workstations, build servers, and CI/CD runners. Identify any packages published by aliases mr.4nd3r50n, ce-rwb, t-in-one, or vpmdhaj, or any packages matching known impersonated scopes (SberPay-related naming). Immediately revoke and rotate any AWS IAM credentials, HashiCorp Vault tokens, and CI/CD secrets (GitHub Actions,

Jenkins, GitLab CI) accessible from affected systems. Apply NIST IR-4 (Incident Handling) procedures to scope the blast radius before allowing affected pipelines to resume.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: On each developer workstation and build node, run: `npm ls --json --all 2>/dev/null | python3 -c "import sys,json; pkgs=json.load(sys.stdin); [print(k) for k in pkgs.get('dependencies',{}).keys()]" | grep -Ei 'sberpay|sber|mr4nd3r50n|ce-rwb|t-in-one|vpmdhaj'`. For AWS credential rotation without enterprise tooling, use AWS CLI: `aws iam list-access-keys --user-name ` then `aws iam delete-access-key --access-key-id ` followed by `aws iam create-access-key`. For CI/CD secrets, manually cycle tokens in GitHub Settings → Secrets, Jenkins Credentials Store, or GitLab CI/CD Variables and audit which pipelines ran during May 25-31 using `git log --after=2026-05-25 --before=2026-05-31` to scope affected jobs.

Evidence: Before rotating or revoking credentials, preserve the following: (1) Full contents of `~/.npm/_logs/` on each developer workstation (npm debug logs capture install timestamps, resolved package versions, and any script execution output). (2) `~/.npmrc` and project-level `.npmrc` files showing registry scope mappings that may have allowed public registry resolution for internal scopes. (3) Node.js process history — on Linux, `~/.bash_history` or `~/.zsh_history` filtered for `npm install` invocations; on Windows, PowerShell transcript logs at `$env:USERPROFILE\Documents\PowerShell\` and `%APPDATA%\npm\` directory listings. (4) AWS CloudTrail events from May 25-31 in JSON format before any IAM changes obscure pre-incident state: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetCallerIdentity --start-time 2026-05-25 --end-time 2026-05-31`. (5) CI/CD pipeline run logs (GitHub Actions: `.github/workflows/` run artifacts; Jenkins: `$JENKINS_HOME/jobs//builds//log`; GitLab CI: pipeline job trace archives) covering the May 25-31 window.

Step 2: Detection — Query SIEM and endpoint logs for npm preinstall/postinstall script execution events spawning child processes (node, sh, bash, cmd) or making outbound HTTP/S connections from build agents. Search package-lock.json and npm audit logs for the 47 known malicious package names (obtain full IOC list from Microsoft Security Blog source). Enable NIST AU-2 (Event Logging) and CIS 8.2 (Collect Audit Logs) on all CI/CD nodes if not already active. Check AWS CloudTrail for GetCallerIdentity, AssumeRole, and ListBuckets calls originating from unexpected EC2 instance metadata service (IMDS) queries or developer workstation IPs. Alert on environment variable enumeration patterns consistent with T1082 and T1119.

NIST Phase: Detection Analysis

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

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

Compensating: Without a SIEM, deploy Sysmon on Windows build agents using SwiftOnSecurity's config (https://github.com/SwiftOnSecurity/sysmon-config) and filter Event ID 1 (Process Create) for parent processes matching `node.exe` or `npm.cmd` spawning `cmd.exe`, `powershell.exe`, `sh`, or `bash`. On Linux CI nodes, enable auditd with rule: `-a always,exit -F arch=b64 -S execve -F ppid=$(pgrep -o npm) -k npm_child_exec`. For package-lock.json scanning across a repo, run: `find . -name 'package-lock.json' | xargs python3 -c "import sys,json; [print(f) or [print(' '+k) for k in json.load(open(f)).get('packages',{}).keys() if any(x in k for x in ['sberpay','sber','mr4nd3r50n','ce-rwb','t-in-one','vpmdhaj'])] for f in sys.argv[1:]"`. For AWS CloudTrail without a SIEM, run: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetCallerIdentity | jq '.Events[] | {time: .EventTime, user: .Username, source: .CloudTrailEvent | fromjson | .sourceIPAddress}'` and flag any developer workstation IPs or unexpected IMDS source addresses (169.254.169.254).

Evidence: Capture before SIEM rule changes alter baseline: (1) Sysmon Event ID 1 logs or auditd execve records showing `node` or `npm` as parent process of any shell, curl, wget, or PowerShell invocation — these represent the preinstall/postinstall hooks used by Campaign 1 (reconnaissance) and Campaign 2 (credential theft) executing at install time. (2) Sysmon Event ID 3 (Network Connection) or `ss -tnp` / `netstat -tnp` captures from build agents showing outbound connections from `node.exe` or `node` processes during the May 25-31 window — Campaign 2

exfiltrated credentials over HTTP/S at install time. (3) Environment variable snapshots: on Linux, `/proc//environ` contents captured during or immediately after a suspect install; on Windows, `Get-Process node | ForEach {[System.Diagnostics.Process]::GetProcessById($_.Id).StartInfo.EnvironmentVariables}` — Campaign 2 specifically targeted `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `VAULT_TOKEN`, `GITHUB_TOKEN`, and `CI`-prefixed variables. (4) AWS CloudTrail `GetCallerIdentity` events — this specific API call is a fingerprint of Campaign 2's AWS credential validation step, used to confirm stolen keys are live before exfiltration. (5) Full `package-lock.json` files from all projects that ran `npm install` during May 25-31, preserving resolved registry URLs which will show whether packages resolved to the public registry instead of a private one (dependency confusion indicator).

Step 3: Eradication — Remove all identified malicious packages from node_modules directories, package.json, and package-lock.json across affected systems. Rotate all credentials confirmed or suspected to have been accessible during any malicious install event: AWS access keys (IAM), HashiCorp Vault tokens, CI/CD pipeline secrets, and any .npmrc tokens. Enforce npm package integrity verification using lockfile integrity hashes (npm ci over npm install) per CWE-494 remediation guidance. Apply CIS 4.6 (Securely Manage Enterprise Assets and Software) by locking internal package scope resolution to a private registry (Artifactory, Nexus, or AWS CodeArtifact) with explicit scope mapping to prevent dependency confusion. Disable or restrict npm lifecycle script execution in CI/CD environments using --ignore-scripts flag where feasible.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST CM-7 (Least Functionality), NIST AC-3 (Access Enforcement), NIST AC-4 (Information Flow Enforcement), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 2.3 (Address Unauthorized Software)

Compensating: For node_modules removal and rebuild on affected systems: `find /path/to/projects -name 'node_modules' -type d -prune -exec rm -rf {} +` then rebuild using `npm ci` (which enforces lockfile integrity) instead of `npm install`. To enforce `--ignore-scripts` globally in CI/CD without enterprise tooling, add to the runner's `.npmrc`: `ignore-scripts=true`. For private registry scope enforcement without Artifactory/Nexus, use AWS CodeArtifact (free tier) and configure `.npmrc` with: `@your-scope:registry=https://your-domain.codeartifact.region.amazonaws.com/npm/your-repo/` — this ensures any package under your internal scope resolves to your private registry and cannot be hijacked via the public registry using dependency confusion. To verify no malicious packages remain, run: `npm ls --depth=Infinity 2>/dev/null | grep -Ei 'sberpay|sber|mr4nd3r50n|ce-rwb|t-in-one|vpmdhaj'` across all project roots. For HashiCorp Vault token rotation, run: `vault token revoke -self` on compromised tokens and reissue via `vault token create -policy= -ttl=`.

Evidence: Before wiping node_modules, preserve forensic copies: (1) Full `node_modules/.package-lock.json` (internal lockfile written by npm v7+) — this records the exact resolved version, registry URL, and integrity hash of every installed package, providing proof of which malicious package version was present. (2) Any files written to disk by malicious postinstall scripts — Campaign 1 (reconnaissance) payloads typically write system profile data to temp directories; search `$TMPDIR`, `/tmp/`, and `%TEMP%` for JSON or base64-encoded files created by `node` processes between May 25-31. (3) `~/.npmrc` and project `.npmrc` files showing the absence of explicit scope-to-registry mappings (the root cause enabling dependency confusion for Campaign 1). (4) HashiCorp Vault audit log entries (`vault audit list` → typically at `/var/log/vault/audit.log`) showing any token usage events between May 25-31, specifically `auth` events with the compromised token accessor IDs. (5) CI/CD secrets access logs: GitHub Actions `Settings → Security → Secret scanning alerts`; Jenkins credentials audit log at `$JENKINS_HOME/audit.log` (requires Audit Trail plugin); GitLab CI audit events API at `/api/v4/audit_events?entity_type=Project`.

Step 4: Recovery — Validate that all rotated credentials are functional and that no residual access remains on compromised IAM roles or Vault paths. Re-run builds from clean environments with verified package manifests. Monitor AWS CloudTrail, Vault audit logs, and CI/CD pipeline logs for 30 days post-remediation for anomalous access patterns consistent with delayed exploitation using profiled environment data from Campaign 1. Confirm private registry scope resolution is enforced and test dependency confusion controls

using a benign internal package name against the public registry. Apply NIST SI-4 (System Monitoring) to maintain elevated detection posture on build infrastructure.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-4 (System Monitoring), NIST CP-10 (System Recovery and Reconstitution), NIST AU-12 (Audit Record Generation), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 6.3 (Require MFA for Externally-Exposed Applications)

Compensating: To validate no residual IAM access exists, run: `aws iam simulate-principal-policy --policy-source-arn arn:aws:iam:::user/ --action-names sts:AssumeRole s3:ListBuckets` for each affected identity and confirm expected denials on sensitive resources. For dependency confusion control validation without a commercial tool, publish a scoped test package (e.g., `@your-internal-scope/confusion-test-do-not-install`) to the public npm registry with a higher version number than your internal one, then run `npm install @your-internal-scope/confusion-test-do-not-install --dry-run` from a clean environment and verify it resolves to your private registry. For 30-day elevated monitoring on build nodes without a SIEM, configure a daily cron job: `0 6 * * * aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetCallerIdentity --start-time $(date -d '24 hours ago' --iso-8601=seconds) | jq '.Events[] | select(.Username != "expected-ci-user")' >> /var/log/ir-monitoring.log` and review daily.

Evidence: During recovery validation, collect and retain: (1) AWS CloudTrail event history for newly rotated IAM key IDs — any `GetCallerIdentity` or `AssumeRole` call using the new key IDs from an unexpected source IP within 24 hours of rotation indicates a threat actor retained a copy of credentials or has persistence via an IAM role not yet rotated. (2) Vault audit log entries showing token accessor activity post-rotation — any access using revoked accessor IDs indicates Vault seal/unseal or audit log integrity issues. (3) `npm ci` build output logs from clean environment rebuilds — these will show the exact package resolution chain including registry source URLs, confirming private registry enforcement is working. (4) Network flow logs (AWS VPC Flow Logs or Linux `conntrack` output) from build agents for 30 days, specifically watching for outbound connections to the C2 infrastructure used by Campaign 1/2 (obtain C2 IP/domain IOCs from the Microsoft Security Blog advisory). (5) CI/CD pipeline artifact hashes (SHA-256 of build outputs) compared between pre-incident and post-recovery builds to detect any trojanized build artifacts that may have been introduced by Campaign 2 packages executing at build time.

Step 5: Post-Incident — Conduct a supply chain security review against NIST SP 800-161r1 (Cyber Supply Chain Risk Management). Implement software composition analysis (SCA) tooling in CI/CD pipelines to detect malicious or untrusted packages before install. Enforce CIS 2.1 (Establish and Maintain a Software Inventory) and CIS 2.3 (Address Unauthorized Software) for npm dependencies. Establish a private npm registry with allowlist-based scope controls (AC-3, AC-4) to eliminate public registry dependency confusion risk. Review and restrict environment variable exposure in CI/CD pipeline configurations to limit credential harvest surface (AC-6, Least Privilege). Document lessons learned and update software supply chain risk in the organizational risk register.

NIST Phase: Post Incident

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

Controls: NIST AC-3 (Access Enforcement), NIST AC-4 (Information Flow Enforcement), NIST AC-6 (Least Privilege), NIST RA-3 (Risk Assessment), NIST SA-12 (Supply Chain Protection), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.3 (Address Unauthorized Software)

Compensating: For SCA without a commercial tool (Snyk, Mend, etc.), integrate `npm audit --audit-level=high` and `npx better-npm-audit` into CI/CD pipelines as a blocking pre-install step. For dependency inventory without a paid SBOM tool, run: `npm ls --json --all > sbom-$(date +%Y%m%d).json` on each project post-build and store artifacts for 90 days. To restrict environment variable exposure in GitHub Actions without enterprise secrets management, scope secrets to specific environments (`environment: production`) and use `GITHUB_TOKEN` permission minimization in workflow YAML: `permissions: contents: read`. For CI/CD pipeline secret segmentation in Jenkins, use the Credentials Binding Plugin to scope credentials to specific pipeline steps and prevent their availability as environment variables during the `npm install` phase. Write a Sigma rule to detect future dependency confusion attempts: match on process creation where parent is `npm` or `node`, child is `curl`/`wget`/`Invoke-WebRequest`, and network destination is not

your private registry FQDN.

Evidence: For lessons-learned documentation, preserve and reference: (1) The complete timeline of `npm install` events from May 25-31 reconstructed from npm debug logs, CI/CD run histories, and endpoint process logs — this is the authoritative record of exposure scope for the risk register update. (2) The full list of environment variables accessible in each affected CI/CD pipeline context during the incident window, documented from pipeline configuration files (`.github/workflows/*.yml`, `Jenkinsfile`, `.gitlab-ci.yml`) — this quantifies the credential harvest surface that Campaign 2 could have accessed and justifies the AC-6 remediation. (3) Dependency confusion test results from recovery validation (Step 4) demonstrating whether your private registry controls would have prevented Campaign 1's attack vector before this incident. (4) AWS IAM Access Analyzer findings report (`aws accessanalyzer list-findings`) run post-rotation to identify any IAM roles with overly broad trust policies that could have been exploited with stolen STS tokens during the incident window. (5) A post-incident package manifest diff comparing `package-lock.json` files before and after the incident across all affected repositories, generated with `git diff HEAD -- package-lock.json` — this provides the authoritative list of packages that changed and supports supply chain risk assessment under NIST SP 800-161r1.

Detection Guidance

Primary detection surface is npm install activity on developer workstations, build agents, and CI/CD runners.

Lifecycle Hook Abuse (T1543, T1053, T1059.007): Alert on npm processes spawning child shells (sh, bash, cmd.exe, powershell) or network connections during or immediately after package installation. In Endpoint Detection logs, search for parent process 'node' or 'npm' with child processes making DNS lookups or TCP connections to non-internal hosts.

Credential Access (T1552.001, T1552.005, T1552.007, T1528): In AWS CloudTrail, query for: GetCallerIdentity from unexpected source IPs; IMDSv1 metadata endpoint access (169.254.169.254) from developer hosts; sudden AssumeRole or CreateAccessKey events. In HashiCorp Vault audit logs, flag token reads from hosts not in the authorized build fleet.

Package Identification: Cross-reference installed packages against the full IOC list from the Microsoft Security Blog (May 28-29, 2026 posts). Threat actor npm aliases: mr.4nd3r50n, ce-rwb, t-in-one, vpmdhaj. Look for packages mimicking internal scopes or SberPay-related naming conventions.

Obfuscation Indicators (T1027): Flag JavaScript files in node_modules containing base64-encoded strings, eval() calls on decoded content, or hex-encoded string concatenation, common obfuscation patterns in these campaigns.

Exfiltration (T1071.001): Monitor outbound HTTP/S from build systems to non-allowlisted domains, particularly short-lived or newly registered domains. Apply network flow analysis on CI/CD egress.

Relevant Controls: NIST AU-2 (Event Logging), AU-6 (Audit Record Review), SI-4 (System Monitoring); CIS 8.2 (Collect Audit Logs).

D3FEND Countermeasures: D3-SFA (System File Analysis) for node_modules integrity; D3-LAM (Local Account Monitoring) for credential access on build hosts; D3-UAP (User Account Permissions) to restrict CI/CD secret access scope.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	See Microsoft Security Blog (May 28-29, 2026) for full exfiltration domain list	Exfiltration endpoints used by Campaign 2 typosquatting packages to receive stolen AWS credentials and CI/CD secrets	HIGH
HASH	See Microsoft Security Blog (May 28-29, 2026) for package file hashes	SHA hashes for the 47 malicious npm packages across both campaigns	HIGH
URL	npm alias: mr.4nd3r50n	Threat actor npm publisher alias associated with malicious packages in both campaigns	HIGH
URL	npm alias: ce-rwb	Threat actor npm publisher alias associated with malicious packages in both campaigns	HIGH
URL	npm alias: t-in-one	Threat actor npm publisher alias associated with malicious packages in both campaigns	HIGH
URL	npm alias: vpmdhaj	Threat actor npm publisher alias associated with malicious packages in both campaigns	HIGH
URL	169.254.169.254	AWS EC2 Instance Metadata Service (IMDS) endpoint abused by Campaign 2 packages to harvest cloud credentials (T1552.005)	HIGH

Framework Mappings

MITRE-ATTACK

- **T1053** — Scheduled Task/Job
- **T1528** — Steal Application Access Token
- **T1119** — Automated Collection
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.005** — Cloud Instance Metadata API
- **T1082** — System Information Discovery
- **T1543** — Create or Modify System Process
- **T1552.007** — Container API
- **T1059.007** — JavaScript
- **T1083** — File and Directory Discovery
- **T1036.001** — Invalid Code Signature
- **T1059.006** — Python
- **T1195** — Supply Chain Compromise

- **T1027** — Obfuscated Files or Information
- **T1071.001** — Web Protocols
- **T1552.001** — Credentials In Files

NIST-800-53R5

- **AC-3** — Access Enforcement
- **CM-7** — Least Functionality
- **AC-6** — Least Privilege
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **SA-9** — External System Services
- **SR-2** — Supply Chain Risk Management Plan
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **SI-3** — Malicious Code Protection
- **CM-3** — Configuration Change Control
- **SC-28** — Protection of Information at Rest

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures
- **A01:2021** — Broken Access Control

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(a)(1)** — Access Control
- **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.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
T1053	Scheduled Task/Job	Execution
T1528	Steal Application Access Token	Credential-Access
T1119	Automated Collection	Collection
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.005	Cloud Instance Metadata API	Credential-Access
T1082	System Information Discovery	Discovery
T1543	Create or Modify System Process	Persistence
T1552.007	Container API	Credential-Access
T1059.007	JavaScript	Execution
T1083	File and Directory Discovery	Discovery
T1036.001	Invalid Code Signature	Defense-Evasion
T1059.006	Python	Execution
T1195	Supply Chain Compromise	Initial-Access
T1027	Obfuscated Files or Information	Defense-Evasion
T1071.001	Web Protocols	Command-And-Control
T1552.001	Credentials In Files	Credential-Access

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://thehackernews.com/2026/05/malicious-sicoob-nuget-steals-ban...	T3
	https://www.microsoft.com/en-us/security/blog/2026/05/28/typosquatt...	T1

Source	URL	Tier
Widespread Supply Chain Compromise Impacting npm Ecosystem	https://www.cisa.gov/news-events/alerts/2025/09/23/widespread-suppl...	T1

Generated 2026-05-30 06:22 UTC by TJS Security Command Center