INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-14 18:50 UTC

node-ipc Supply Chain Compromise: Credential-Harvesting Backdoor Targets 90 Secret Categories Across Cloud and Dev Environments

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0313
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	node-ipc npm package versions 9.1.6, 9.2.3, 12.0.1
Published	2026-05-14T13:22:43
Discovery Source	Rss

Executive Summary

Three malicious versions of the node-ipc npm package (9.1.6, 9.2.3, 12.0.1) were published via a compromised or unauthorized npm account and contain a hidden credential-harvesting backdoor targeting 90 categories of secrets, including AWS, Azure, Google Cloud, GitHub, Kubernetes, Terraform, and SSH credentials. Any organization using these versions directly or via downstream dependencies such as vue-cli should treat affected systems as potentially compromised and immediately audit their dependency trees. The business risk is severe: stolen cloud and infrastructure credentials can enable full environment takeover, data exfiltration, ransomware deployment, and regulatory breach obligations.

Technical Analysis

Three versions of the node-ipc npm package (9.1.6, 9.2.3, 12.0.1) were published via a compromised or unauthorized npm account ('atiertant') and contain an obfuscated, self-executing backdoor embedded directly in the package's core module. This placement bypasses standard npm supply chain defenses that monitor preinstall/postinstall lifecycle hooks. The payload targets 90 secret categories including AWS IAM credentials, GCP service account keys, Azure tokens, GitHub CLI tokens, Kubernetes service account tokens, Terraform state credentials, SSH private keys, and AI tooling configurations (Claude AI, Kiro IDE). Exfiltration occurs over dual channels: HTTPS and DNS TXT record queries (T1041, T1048.002, T1071.004) directed at a suspected Azure-themed C2 domain. Version 12.0.1 is reported to include conditional activation logic, suggesting targeting

of specific environments, consistent with a targeted, pre-planned attack (T1195.001) rather than opportunistic compromise. The legitimate node-ipc package is a transitive dependency of vue-cli and other widely used Node.js tooling, significantly expanding the potential blast radius. Relevant CWEs: CWE-522 (insufficiently protected credentials), CWE-494 (download of code without integrity check), CWE-200 (exposure of sensitive information), CWE-506 (embedded malicious code), CWE-912 (hidden functionality). MITRE techniques include T1552.001, T1552.005, T1027, T1036, T1059.007, T1078, T1078.001, T1083, T1560.001, T1195.001. Note: CVE-2022-23812 (Snyk) documents a separate 2022 node-ipc incident involving geopolitically motivated file destruction; this 2026 campaign is a distinct event and should not be conflated with that prior compromise.

Action Checklist

1. Step 1: Containment, Immediately identify all systems, CI/CD pipelines, and developer workstations where node-ipc versions 9.1.6, 9.2.3, or 12.0.1 are installed directly or as a transitive dependency (e.g., via vue-cli). Isolate affected build environments from production networks and block outbound HTTPS and DNS traffic to unknown or newly observed Azure-themed domains pending investigation.
2. Step 2: Detection, Audit npm lock files (package-lock.json, yarn.lock) and node_modules directories across all repositories, CI/CD runners, and developer machines for the three affected versions. Query DNS and proxy logs for TXT record lookups and HTTPS connections to unrecognized domains from build or Node.js runtime processes. Review environment variable access logs and secrets manager audit trails for unexpected reads of cloud credentials, SSH keys, or API tokens. Check for the npm account 'atiertant' in any package provenance records.
3. Step 3: Eradication, Remove affected versions (9.1.6, 9.2.3, 12.0.1) from all environments. Update node-ipc to a known-clean version, verifying package integrity against npm registry checksums. Rotate all credentials accessible in environments where affected versions executed: AWS IAM keys, GCP service account keys, Azure tokens, GitHub tokens, Kubernetes service account tokens, Terraform credentials, SSH private keys, and any AI tooling API keys. Revoke and reissue rather than reset where possible.
4. Step 4: Recovery, After credential rotation, validate that no unauthorized IAM users, service accounts, SSH authorized keys, or API tokens were created using stolen credentials. Audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for anomalous activity during the window of exposure. Re-run clean dependency installs from trusted registry mirrors with integrity verification enabled before restoring CI/CD pipelines to production use.
5. Step 5: Post-Incident, Review and enforce npm package integrity controls: implement Subresource Integrity checks, lock file enforcement, and private registry proxying with allowlist policies. Add detection coverage for self-executing functions in package core modules and DNS TXT exfiltration patterns. Evaluate whether software composition analysis (SCA) tooling would have caught the malicious versions at install time. Document the transitive dependency exposure path via vue-cli as a control gap and assess whether critical pipelines carry unnecessary transitive dependencies.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO, legal counsel, and cloud platform owners immediately if CloudTrail, GCP Audit Logs, or Azure Activity Log show any API calls, resource creation, or data access events using credentials that were present in affected environments during the exposure window, as this confirms credential weaponization and may trigger breach notification obligations under GDPR, CCPA, or SOC 2 incident response SLAs.
Recovery Notes	After credential rotation, maintain elevated monitoring on all newly issued cloud credentials, SSH keys, and CI/CD tokens for a minimum of 30 days, alerting on any usage patterns inconsistent with expected pipeline or user behavior using CloudTrail Insights or equivalent. Validate the integrity of any production deployments made from affected CI/CD pipelines during the exposure window, as build artifacts produced while the backdoor was active may themselves contain compromised secrets baked into container images, Terraform state files, or deployment packages. Re-establish trust in the build pipeline only after confirming clean dependency graphs via `npm ci --ignore-scripts` from a private registry mirror with all three malicious version hashes explicitly blocklisted.
Forensic Artifacts	DNS resolver logs or network PCAP filtered for TXT record queries (dns.qry.type == 16) originating from Node.js or npm process IPs during the exposure window — the node-ipc backdoor used DNS TXT records as an exfiltration channel for harvested credentials | Preserved node_modules/node-ipc/ directory from affected hosts with SHA-256 hashes of all files, specifically the package entry point and any files under services/ or lib/ subdirectories containing the credential-harvesting payload code | AWS CloudTrail GetSecretValue, GetParameter, AssumeRole, and CreateUser events; GCP secretmanager.versions.access and iam.serviceAccounts.create events; Azure Key Vault SecretGet and role assignment write events — all scoped to the exposure window defined by the first install of the malicious version | CI/CD pipeline execution logs (GitHub Actions workflow logs, Jenkins build console output, GitLab CI job traces) capturing the environment variable state and npm install output during runs that executed the affected node-ipc versions, as these logs may show the process startup environment including injected secrets | npm package provenance records and registry audit entries confirming the 'atiertant' account's publication of versions 9.1.6, 9.2.3, and 12.0.1, plus package-lock.json and yarn.lock snapshots from all affected repositories documenting the transitive dependency resolution path from vue-cli to the malicious node-ipc versions

Per-Action IR Details

Step 1: Containment — Immediately identify all systems, CI/CD pipelines, and developer workstations where node-ipc versions 9.1.6, 9.2.3, or 12.0.1 are installed directly or as a transitive dependency (e.g., via vue-cli). Isolate affected build environments from production networks and block outbound HTTPS and DNS traffic to unknown or newly observed Azure-themed domains pending investigation.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory)

Compensating: Run `npm ls node-ipc 2>/dev/null | grep -E '9\.1\.6|9\.2\.3|12\.0\.1'` recursively across all project directories and CI runner workspaces. On Linux/macOS use `find / -path '*/node_modules/node-ipc/package.json' -exec grep -l '"version": "9\.1\.6\|9\.2\.3\|12\.0\.1"' {} \;`. Block outbound DNS and HTTPS from build hosts immediately using host-based firewall rules: `iptables -A OUTPUT -p tcp --dport 443 -j DROP` and `iptables -A OUTPUT -p udp --dport 53 -j DROP` on affected runners until investigation completes.

Evidence: Before isolating, capture full network connection state from all affected build hosts with `ss -tulpn` and `netstat -anp` to record any active outbound connections initiated by node processes. Dump DNS resolver cache

(`systemd-resolve --statistics` or `ipconfig /displaydns` on Windows) to capture any Azure-themed domains already resolved. Preserve the exact node_modules/node-ipc directory tree with file hashes (`sha256sum node_modules/node-ipc/**/*`) before any remediation alters it.

Step 2: Detection — Audit npm lock files (package-lock.json, yarn.lock) and node_modules directories across all repositories, CI/CD runners, and developer machines for the three affected versions. Query DNS and proxy logs for TXT record lookups and HTTPS connections to unrecognized domains from build or Node.js runtime processes. Review environment variable access logs and secrets manager audit trails for unexpected reads of cloud credentials, SSH keys, or API tokens. Check for the npm account 'atiertant' in any package provenance records.

NIST Phase: Detection Analysis

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

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

Compensating: Query DNS logs for TXT record lookups originating from Node.js or npm processes using Zeek or tcpdump: `tcpdump -nn -i any 'udp port 53 and (host )' -w dns_capture.pcap`, then parse with `tshark -r dns_capture.pcap -Y 'dns.qry.type == 16' -T fields -e dns.qry.name` (type 16 = TXT). For secrets manager access, query AWS CloudTrail with: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=GetSecretValue --start-time `. Search all lock files recursively: `grep -r 'node-ipc' **/package-lock.json **/yarn.lock | grep -E '9\.1\.6|9\.2\.3|12\.0\.1'`. Check npm provenance with `npm view node-ipc@9.1.6 _npmUser` to confirm the 'atiertant' account association.

Evidence: Retrieve DNS query logs from the resolver or network tap covering the full exposure window and filter for TXT record queries from build host IPs — the node-ipc backdoor uses DNS TXT records as an exfiltration or C2 channel. Pull AWS CloudTrail GetSecretValue, GetParameter (SSM), and AssumeRole events; GCP Audit Log `secretmanager.versions.access` events; and Azure Key Vault diagnostic log `SecretGet` operations scoped to the exposure window. Capture environment variable enumeration artifacts: on Linux, `/proc//environ` snapshots if the process was still running, or audit logs from auditd rule `-a always,exit -F arch=b64 -S execve` showing node process invocations with environment inheritance.

Step 3: Eradication — Remove affected versions (9.1.6, 9.2.3, 12.0.1) from all environments. Update node-ipc to a known-clean version, verifying package integrity against npm registry checksums. Rotate all credentials accessible in environments where affected versions executed: AWS IAM keys, GCP service account keys, Azure tokens, GitHub tokens, Kubernetes service account tokens, Terraform credentials, SSH private keys, and any AI tooling API keys. Revoke and reissue rather than reset where possible.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST IA-5 (Authenticator Management), NIST CM-7 (Least Functionality), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 7.4 (Perform Automated Application Patch Management), CIS 5.2 (Use Unique Passwords)

Compensating: Verify npm package integrity before reinstalling: `npm view node-ipc@ dist.integrity` and compare the shasum against the locally cached tarball with `sha512sum ~/.npm/_cacache/`. For AWS, rotate IAM access keys via `aws iam create-access-key --user-name ` then immediately `aws iam delete-access-key --access-key-id `. For GitHub, invalidate all PATs and OAuth tokens via the GitHub API: `curl -X DELETE -H 'Authorization: token ' https://api.github.com/installation/token`. For SSH, regenerate host and user keys and replace all `~/.ssh/authorized_keys` entries that existed during the exposure window. For Kubernetes, rotate service account tokens with `kubectl delete secret -n ` to force reissuance.

Evidence: Before package removal, preserve a forensic copy of the malicious node-ipc module files — specifically `node_modules/node-ipc/services/` and the main entry point defined in its `package.json` `main` field — as these contain the backdoor payload code. Hash all files with SHA-256 for chain of custody. Document the exact dependency resolution path (e.g., `vue-cli → @vue/cli-service → node-ipc@9.1.6`) from lock files before deletion. Capture a list of

all environment variables present in the build environment at time of infection via CI/CD runner logs or pipeline execution records, which establishes the full scope of potentially harvested secrets.

Step 4: Recovery — After credential rotation, validate that no unauthorized IAM users, service accounts, SSH authorized_keys, or API tokens were created using stolen credentials. Audit cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for anomalous activity during the window of exposure. Re-run clean dependency installs from trusted registry mirrors with integrity verification enabled before restoring CI/CD pipelines to production use.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AC-2 (Account Management), NIST CA-7 (Continuous Monitoring), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 5.3 (Disable Dormant Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Query AWS CloudTrail for account creation and privilege escalation events during the exposure window: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=CreateUser` and `AttributeValue=AttachUserPolicy`. For GCP, run `gcloud logging read 'protoPayload.methodName=google.iam.admin.v1.CreateServiceAccount' --freshness=`. For Azure, query Activity Log for `Microsoft.Authorization/roleAssignments/write` events. Check GitHub audit log for new OAuth app authorizations, SSH key additions, or repository access grants during the window via `https://api.github.com/orgs//audit-log?phrase=action:oauth_access`. Before restoring pipelines, re-run `npm ci --ignore-scripts` with `npm config set registry https://` to prevent install-time script execution.

Evidence: Pull the full IAM diff from each cloud provider for the exposure window: AWS IAM credential report (`aws iam generate-credential-report && aws iam get-credential-report`), GCP IAM policy export (`gcloud projects get-iam-policy `), and Azure role assignment history from the Activity Log. Collect SSH `authorized_keys` file modification timestamps across all affected hosts using `find /home -name authorized_keys -newer ` and compare against known-good baselines. Preserve Kubernetes RBAC audit logs showing any ClusterRoleBinding or RoleBinding creation events during the exposure window from the Kubernetes API server audit log.

Step 5: Post-Incident — Review and enforce npm package integrity controls: implement Subresource Integrity checks, lock file enforcement, and private registry proxying with allowlist policies. Add detection coverage for self-executing functions in package core modules and DNS TXT exfiltration patterns. Evaluate whether software composition analysis (SCA) tooling would have caught the malicious versions at install time. Document the transitive dependency exposure path via vue-cli as a control gap and assess whether critical pipelines carry unnecessary transitive dependencies.

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity (Lessons Learned)

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST SI-5 (Security Alerts, Advisories, and Directives), NIST IR-8 (Incident Response Plan), 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 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Deploy YARA rules scanning node_modules directories for self-invoking function patterns characteristic of this backdoor (e.g., `(function(){...})()` blocks in package entry points that contain base64-encoded strings or DNS query logic). Write a Sigma rule targeting DNS TXT query events from Node.js processes: match on `process.name: 'node'` AND `dns.question.type: 'TXT'`. Enforce lock file integrity in CI by adding `npm ci` (not `npm install`) with the `--ignore-scripts` flag to all pipeline definitions. Use `npm audit signatures` (npm CLI v8.7+) to verify package signature provenance before install. Deploy osquery with a scheduled query against `npm_packages` table to continuously inventory installed npm packages and alert on the three malicious version hashes.

Evidence: Compile the full transitive dependency graph showing how node-ipc entered each affected environment (direct install vs. vue-cli transitive path) using `npm ls node-ipc --all` output preserved from pre-remediation snapshots. Retain all DNS TXT query logs, secrets manager audit trails, and cloud provider access logs for the full exposure window as post-incident documentation. Archive the malicious package files and computed SHA-256 hashes as threat

intelligence artifacts for sharing with your SCA vendor and for future YARA/Sigma rule development. Document the npm account 'atiertant' as a threat indicator in your internal threat intelligence platform for monitoring against future package publications.

Detection Guidance

Audit all npm dependency trees for node-ipc versions 9.1.6, 9.2.3, or 12.0.1: run 'npm ls node-ipc' or 'yarn why node-ipc' across repositories and CI runners. In DNS logs, query for TXT record lookups originating from Node.js processes or build agents, particularly to domains with Azure-themed names not in your approved domain inventory. In proxy/firewall logs, look for HTTPS POST or GET requests from npm install processes or Node.js runtimes to unrecognized external hosts. In secrets manager and cloud provider audit logs (CloudTrail, GCP Audit Logs, Azure Activity Log), flag reads of IAM credentials, service account keys, or API tokens occurring during or shortly after build pipeline execution. Technical IoC: the malicious payload is embedded as a self-executing function in the package's main entry point (node-ipc/lib/index.js or equivalent). Static code review or file integrity monitoring should flag obfuscated function definitions that execute on module load without explicit require() or function call. Behavioral indicator: the payload is a self-executing function in the package core module, static analysis or file integrity monitoring on node_modules may surface obfuscated code blocks in node-ipc's main entry file. Hash-gating in version 12.0.1 means the payload may not fire in all environments; absence of observed exfiltration does not confirm clean execution.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`typosquatted Azure-themed C2 domain (specific domain not confirmed in available sources)`	Command-and-control domain used for dual-channel exfiltration via HTTPS and DNS TXT record queries; exact domain not disclosed in available T3 sources — monitor for Azure-themed domains not in approved inventory	MEDIUM
HASH	`SHA-256 hash-gating value in version 12.0.1 (specific hash not confirmed in available sources)`	Version 12.0.1 activates payload only when entry point hash matches a pre-targeted value; specific hash not disclosed in available sources	LOW
URL	`npm package: node-ipc version 9.1.6`	Confirmed malicious version containing credential-harvesting backdoor	HIGH
URL	`npm package: node-ipc version 9.2.3`	Confirmed malicious version containing credential-harvesting backdoor	HIGH
URL	`npm package: node-ipc version 12.0.1`	Confirmed malicious version containing credential-harvesting backdoor with SHA-256 hash-gating for targeted activation	HIGH

Framework Mappings

MITRE-ATTACK

- **T1041** — Exfiltration Over C2 Channel
- **T1048.002** — Exfiltration Over Asymmetric Encrypted Non-C2 Protocol
- **T1078** — Valid Accounts
- **T1552.005** — Cloud Instance Metadata API
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1059.007** — JavaScript
- **T1071.004** — DNS
- **T1027** — Obfuscated Files or Information
- **T1036** — Masquerading
- **T1560.001** — Archive via Utility
- **T1083** — File and Directory Discovery
- **T1552.001** — Credentials In Files
- **T1078.001** — Default Accounts

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **SI-3** — Malicious Code Protection
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **AC-3** — Access Enforcement
- **SC-28** — Protection of Information at Rest
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

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

CIS-V8

- **5.2** — Use Unique Passwords

- **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
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **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
- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1041	Exfiltration Over C2 Channel	Exfiltration
T1048.002	Exfiltration Over Asymmetric Encrypted Non-C2 Protocol	Exfiltration
T1078	Valid Accounts	Defense-Evasion
T1552.005	Cloud Instance Metadata API	Credential-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1059.007	JavaScript	Execution
T1071.004	DNS	Command-And-Control
T1027	Obfuscated Files or Information	Defense-Evasion
T1036	Masquerading	Defense-Evasion

Technique ID	Technique Name	Tactic
T1560.001	Archive via Utility	Collection
T1083	File and Directory Discovery	Discovery
T1552.001	Credentials In Files	Credential-Access
T1078.001	Default Accounts	Defense-Evasion

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/stealer-backdoor-found-in-3-node-...	T3
Major security issue with npm. Stop using npm immediately. - Reddit	https://www.reddit.com/r/ClaudeCode/comments/1tbmcya/major_security...	T3
Malicious npm Packages Backdoor Claude Code Sessions - SafeDep	https://safedep.io/malicious-npm-packages-claude-code-hooks	T3
Malicious Package in node-ipc | CVE-2022-23812 | Snyk	https://security.snyk.io/vuln/SNYK-JS-NODEIPC-2426370	T3
!!!!!!!!!!! Please do something to warn USERS besides publishing ...	https://github.com/vuejs/vue-cli/issues/7054	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 2026-05-14 18:50 UTC by TJS Security Command Center