INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-02 06:31 UTC

Miasma Malware Hits Red Hat npm Namespace: GitHub OIDC Abuse Enables Mass Credential Theft Across 32 Packages

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0390
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Red Hat @redhat-cloud-services npm namespace (32 packages, 96 versions); downstream exposure in environments using AWS, Google Cloud, Azure, HashiCorp Vault, Kubernetes, npm, PyPI, Docker, and GitHub Actions where affected packages were installed
Published	2026-06-01T17:38:29
Discovery Source	Rss

Executive Summary

Attackers compromised a Red Hat employee's GitHub account and used it to inject credential-stealing malware into 32 npm packages under the official '@redhat-cloud-services' namespace, collectively downloaded approximately 117,000 times per week. The malware, tracked as Miasma, targets cloud provider credentials, CI/CD pipeline tokens, SSH keys, and developer secrets, meaning any build environment that installed affected packages may have already exfiltrated sensitive access material. Organizations using these packages in AWS, Google Cloud, Azure, Kubernetes, or GitHub Actions pipelines face potential unauthorized cloud account access, data exfiltration, and lateral movement across their entire infrastructure.

Technical Analysis

Between an unspecified date and public disclosure, threat actors operating the Miasma supply chain campaign compromised a Red Hat employee's GitHub account. They leveraged GitHub's OIDC trusted publishing integration to authenticate directly with npm's registry, bypassing token-based controls and standard secret-scanning defenses entirely. This allowed unauthorized publication of backdoored versions across 32 packages and 96 versions in the '@redhat-cloud-services' npm namespace. The Miasma payload harvests cloud credentials (AWS, GCP, Azure), CI/CD tokens, SSH private keys (T1552.004), environment-stored plaintext secrets (T1552.001, CWE-312, CWE-522), and exfiltrates via external channels (T1567). The attack

chain maps to T1195.001 and T1195.002 (Supply Chain Compromise, software and dependency), T1554 (Compromise Software Supply Chain), T1528 (Steal Application Access Token), T1078.004 (Valid Accounts, Cloud), T1059.007 (JavaScript execution), T1070.003 (log history clearing), and T1650 (Acquire Access). CWE-506 (embedded malicious code) and CWE-829 (inclusion of untrusted functionality) describe the package-level weaknesses. The OIDC abuse technique is particularly significant: it subverts a control widely recommended as a hardening measure, meaning organizations that rotated npm tokens and enabled trusted publishing may have believed they were protected. 309 GitHub repositories are confirmed compromised in the broader Miasma campaign. No CVE has been assigned. No patched versions exist, affected packages must be removed and replaced. The Microsoft Security Blog (T1 source, 2026-05-28) documents the broader typosquatting and secret-theft campaign context.

Action Checklist

1. Step 1: Containment. Immediately audit all package.json, package-lock.json, and yarn.lock files across build systems, CI/CD pipelines, and developer workstations for any '@redhat-cloud-services' npm packages. Isolate any environment where affected packages were installed from network access and credential stores. Block the '@redhat-cloud-services' namespace from your internal registry or npm proxy (e.g., Artifactory, Nexus) until Red Hat issues clean verified releases. Do not simply uninstall and continue, treat the environment as compromised until forensic review confirms no exfiltration occurred.
2. Step 2: Detection. Query SIEM and EDR for npm install events involving '@redhat-cloud-services/*' packages across the 96 affected versions. Search CI/CD logs (GitHub Actions, Jenkins, GitLab CI) for pipeline runs that downloaded these packages. Audit cloud provider audit logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for unexpected API calls, new IAM role assumptions, or credential use from unfamiliar sources following build pipeline execution. Look for outbound HTTP/HTTPS connections from build agents to unfamiliar external endpoints (T1567 exfiltration indicator). Review SSH key usage logs for anomalous authentication patterns (NIST AU-6, AU-12). Check for evidence of log clearing in shell history files (T1070.003). Use CIS 8.2 audit log collection baselines to identify gaps in build system logging coverage.
3. Step 3: Eradication. Remove all affected '@redhat-cloud-services' packages from all environments. No patched replacement versions are currently available; do not reinstall from the same namespace until Red Hat confirms supply chain integrity. Rotate all credentials that may have been present in any compromised environment: AWS access keys, GCP service account keys, Azure service principals, HashiCorp Vault tokens, npm publish tokens, PyPI tokens, Docker registry credentials, GitHub personal access tokens, and SSH private keys (NIST IA-5, D3-CRO). Revoke any OIDC trust relationships with npm that are not strictly necessary, and audit remaining OIDC configurations for scope creep (NIST AC-3, AC-6). Disable compromised GitHub account access and enforce MFA on all developer GitHub accounts (CIS 6.3, CIS 6.5, D3-MFA).
4. Step 4: Recovery. Before restoring pipeline operations, verify clean package sources using hash verification against known-good manifests (D3-FMBV). Confirm that rotated credentials have propagated and old credentials are fully invalidated across all services. Re-run CI/CD pipelines in isolated environments and monitor for anomalous outbound network connections before promoting builds to production. Validate cloud provider audit logs show no ongoing unauthorized access using the newly rotated credentials. Re-enable affected pipelines only after confirming clean dependency trees and credential rotation is complete (NIST IR-4, AU-6).

5. Step 5: Post-Incident. Conduct a control gap review focused on two specific failures this attack exposed: (1) OIDC trusted publishing was treated as a security hardening control but provided no protection against account compromise, document your OIDC trust surface and implement additional publisher verification steps; (2) dependency namespace trust was implicit, implement a verified internal mirror or allow-list for approved npm namespaces using a proxy registry, and enforce NIST CM-7 (least functionality) and CIS 2.3 (address unauthorized software) to prevent unapproved packages from reaching build environments. Implement D3-SFA (system file analysis) for build agent credential stores and D3-LAM (local account monitoring) for developer machine anomaly detection. Require MFA on all accounts with npm publish rights (CIS 6.5). Review GitHub Actions OIDC scope grants organization-wide and remove any that grant broader permissions than required (NIST AC-6, AC-5).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and begin breach notification assessment if AWS CloudTrail, GCP Audit Logs, or Azure Activity Logs show any API calls using credentials that were present in a confirmed compromised build environment, or if any of the 32 affected packages are confirmed installed in a pipeline with access to production systems, PII datastores, or regulated data environments — given the 117,000 weekly download count and credential-theft payload, assume exfiltration occurred in any confirmed-install environment until forensics prove otherwise.
Recovery Notes	Do not promote any build to production that was produced by a pipeline environment where affected @redhat-cloud-services packages were installed, even after package removal — the build artifacts themselves may have been backdoored by Miasma at compile/bundle time and should be treated as untrusted. Monitor all newly rotated cloud credentials for a minimum of 30 days post-rotation using CloudTrail/GCP Audit/Azure Activity anomaly baselines, specifically watching for the same API call patterns (AssumeRole, CreateAccessKey, service account key creation) from new source IPs, which would indicate attacker persistence via a credential copy not yet identified. Conduct a follow-up dependency audit at 72 hours and 7 days post-recovery to confirm no re-introduction of affected packages via transitive dependencies or developer workstations that were not initially in scope.

Forensic Artifacts	npm install cache at `~/.npm/_cacache/` and CI runner workspace directories (e.g., `/home/runner/work//` on GitHub Actions) — Miasma's malicious postinstall scripts execute at npm install time and would leave execution traces in the npm debug log at `~/.npm/_logs/` with timestamps correlating to the install event | AWS CloudTrail events for `AssumeRole`, `GetCallerIdentity`, `ListBuckets`, and `CreateAccessKey` originating from CI runner IP addresses in the window following any pipeline run that installed affected packages — Miasma specifically targets AWS credentials and these API calls would appear within seconds of a successful install on a runner with AWS access | Outbound DNS query logs and network flow records from build agents showing connections to non-registry, non-GitHub domains during npm install execution — Miasma's exfiltration mechanism (MITRE T1567) would produce distinctive HTTPS POST traffic to attacker-controlled infrastructure that would appear in pcap or DNS logs captured on the build network segment | Contents of `~/.aws/credentials`, `~/.config/gcloud/application_default_credentials.json`, `~/.kube/config`, and any `.env` files in pipeline workspace directories at the time of compromise — these are the specific credential file paths targeted by supply chain credential-harvesting malware of this type, and file access timestamps on these paths would confirm Miasma execution | GitHub Actions workflow run logs (retained at `https://github.com///actions/runs/`) for all pipeline executions during the affected package publication window — these logs capture the exact npm install output including postinstall script execution, and any Miasma execution would appear as unexpected output lines or process exits in the `npm install` step

Per-Action IR Details

Step 1: Containment — Immediately audit all package.json, package-lock.json, and yarn.lock files across build systems, CI/CD pipelines, and developer workstations for any '@redhat-cloud-services' npm packages. Isolate any environment where affected packages were installed. Block the '@redhat-cloud-services' namespace from your internal registry or npm proxy (e.g., Artifactory, Nexus) until Red Hat issues clean verified releases. Do not simply uninstall and continue — treat the environment as compromised.

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment, Eradication, and Recovery: Choosing a Containment Strategy

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), NIST AC-4 (Information Flow Enforcement), CIS 2.3 (Address Unauthorized Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Run the following one-liner across all developer workstations and CI runners to enumerate affected packages: `find / -name 'package.json' -not -path '*/node_modules/*' 2>/dev/null | xargs grep -l '@redhat-cloud-services'`. For Artifactory or Nexus without a UI block, add a deny rule via their REST API or manually remove the remote proxy cache entry for the '@redhat-cloud-services' scope. For GitHub Actions runners, add a pre-job step that fails the pipeline if any @redhat-cloud-services package appears in the lockfile: `grep -r '@redhat-cloud-services' package-lock.json && exit 1`.

Evidence: Before isolating any environment, preserve a full snapshot of the installed node_modules directory (specifically `node_modules/@redhat-cloud-services/`), the exact contents of package.json, package-lock.json, and yarn.lock as they existed at time of discovery — these files will confirm which of the 96 affected versions were installed and when. Capture the npm install cache at `~/.npm/_cacache/` and any CI runner workspace directories (e.g., `/home/runner/work/` on GitHub-hosted runners) before they are wiped by pipeline cleanup jobs. On developer workstations, capture shell history files (`~/.bash_history`, `~/.zsh_history`) to reconstruct when `npm install` was last executed against affected packages.

Step 2: Detection — Query SIEM and EDR for npm install events involving '@redhat-cloud-services/*' packages across the 96 affected versions. Search CI/CD logs (GitHub Actions, Jenkins, GitLab CI) for pipeline runs that downloaded these packages. Audit cloud provider audit logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) for unexpected API calls, new IAM role assumptions, or credential use from unfamiliar sources following build pipeline execution. Look for outbound HTTP/HTTPS connections from build agents to

unfamiliar external endpoints (T1567 exfiltration indicator). Review SSH key usage logs for anomalous authentication patterns (NIST AU-6, AU-12). Check for evidence of log clearing in shell history files (T1070.003). Use CIS 8.2 audit log collection baselines to identify gaps in build system logging coverage.

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis: Signs of an Incident

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

Compensating: Without a SIEM, use osquery to detect Miasma execution artifacts: `SELECT * FROM process_events WHERE cmdline LIKE '%@redhat-cloud-services%' OR parent_path LIKE '%node%';`. Deploy the following Sigma rule concept manually via grep against CI logs: search GitHub Actions workflow logs for any job that ran `npm install` or `npm ci` after the affected package publication dates and produced outbound connections to non-GitHub, non-npm domains. For AWS CloudTrail without a SIEM, use the AWS CLI: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=AssumeRole --start-time ` to find unexpected role assumptions originating from CI runner IPs. Use Wireshark or `tcpdump -w capture.pcap -i any port 443` on build agents during a controlled re-execution of the pipeline (in an isolated network) to capture Miasma's exfiltration traffic.

Evidence: Capture AWS CloudTrail `AssumeRole`, `GetCallerIdentity`, and `CreateAccessKey` events in the time window following any CI/CD pipeline run that installed affected packages — Miasma specifically targets cloud credentials and would trigger these API calls from build agent IP addresses. In GCP, look for `google.iam.admin.v1.CreateServiceAccountKey` events in Cloud Audit Logs. In Azure, search Activity Logs for `Microsoft.Authorization/roleAssignments/write` events from unfamiliar principals. On the CI runner filesystem, examine `/tmp/`, `/var/tmp/`, and the runner's home directory for staged exfiltration files or encoded blobs. Check `~/.ssh/known_hosts` and `~/.ssh/authorized_keys` on developer machines for newly added entries that postdate the compromised install, which would indicate Miasma leveraged harvested SSH keys.

Step 3: Eradication — Remove all affected '@redhat-cloud-services' packages from all environments. No patched replacement versions are currently available; do not reinstall from the same namespace until Red Hat confirms supply chain integrity. Rotate all credentials that may have been present in any compromised environment: AWS access keys, GCP service account keys, Azure service principals, HashiCorp Vault tokens, npm publish tokens, PyPI tokens, Docker registry credentials, GitHub personal access tokens, and SSH private keys (NIST IA-5, D3-CRO). Revoke any OIDC trust relationships with npm that are not strictly necessary, and audit remaining OIDC configurations for scope creep (NIST AC-3, AC-6). Disable compromised GitHub account access and enforce MFA on all developer GitHub accounts (CIS 6.3, CIS 6.5, D3-MFA).

NIST Phase: Eradication

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

Controls: NIST IA-5 (Authenticator Management), NIST AC-3 (Access Enforcement), NIST AC-6 (Least Privilege), NIST AC-2 (Account Management), CIS 5.3 (Disable Dormant Accounts), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access)

Compensating: For credential rotation without enterprise tooling: AWS — use `aws iam delete-access-key` followed by `aws iam create-access-key` for each affected IAM user, then immediately update all pipeline secrets; GitHub — revoke PATs via `https://github.com/settings/tokens` and audit OAuth app grants at `https://github.com/settings/applications`; HashiCorp Vault — run `vault token revoke -mode=path auth/token/` to mass-revoke tokens scoped to the compromised environment. For OIDC trust audit on GitHub Actions, enumerate all workflows with `permissions: id-token: write` using: `grep -r 'id-token: write' .github/workflows/` across all repositories — each hit is a trust relationship that granted npm publish access and must be reviewed. For SSH key rotation, generate new keys with `ssh-keygen -t ed25519` and remove the compromised public keys from all `~/.ssh/authorized_keys` files and GitHub/GitLab account SSH key settings.

Evidence: Before rotating credentials, document and preserve the full list of credentials in scope for rotation — specifically, capture the output of `aws iam list-access-keys --user-name ` for all IAM users associated with CI/CD pipelines, and `npm token list` to enumerate all npm publish tokens that existed during the exposure window. Preserve GitHub Actions secret names (not values) from affected repository settings to establish the credential rotation audit

trail. Capture the GitHub account's OAuth app authorization list and OIDC trust configurations before disabling the compromised account, as these establish the blast radius of the original account compromise used to inject Miasma.

Step 4: Recovery — Before restoring pipeline operations, verify clean package sources using hash verification against known-good manifests (D3-FMBV). Confirm that rotated credentials have propagated and old credentials are fully invalidated across all services. Re-run CI/CD pipelines in isolated environments and monitor for anomalous outbound network connections before promoting builds to production. Validate cloud provider audit logs show no ongoing unauthorized access using the newly rotated credentials. Re-enable affected pipelines only after confirming clean dependency trees and credential rotation is complete (NIST IR-4, AU-6).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Eradication and Recovery: Recovery

Controls: NIST IR-4 (Incident Handling), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CM-3 (Configuration Change Control), NIST SI-2 (Flaw Remediation), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: For package integrity verification without enterprise tooling, use npm's built-in integrity field: compare the `integrity` hash in your package-lock.json against the hash published in the npm registry for a known-clean version using `npm view @redhat-cloud-services/@ dist.integrity`. For isolated pipeline re-execution, use Docker's `--network=none` flag to run the build container with no external network access and confirm the build succeeds without attempting outbound connections — any Miasma remnant would cause the build to hang or error on its exfiltration attempt. Validate old AWS access keys are fully invalidated by attempting a test API call with the old key after rotation: `aws sts get-caller-identity` should return an `InvalidClientTokenId` error for rotated keys.

Evidence: Before re-enabling production pipelines, capture a baseline network flow log from the isolated test pipeline execution — specifically, record all DNS queries made by the build agent using `tcpdump -i any udp port 53` during the test run. Any DNS resolution to non-npm, non-registry domains during `npm install` is a Miasma indicator of a still-compromised environment. In AWS CloudTrail, confirm that the old access key IDs (preserved from Step 3) show zero successful API calls in the 24 hours following rotation — any successful call would indicate the key was not fully invalidated or was copied to an additional location not yet rotated.

Step 5: Post-Incident — Conduct a control gap review focused on two specific failures this attack exposed: (1) OIDC trusted publishing was treated as a security hardening control but provided no protection against account compromise — document your OIDC trust surface and implement additional publisher verification steps; (2) dependency namespace trust was implicit — implement a verified internal mirror or allow-list for approved npm namespaces using a proxy registry, and enforce NIST CM-7 (least functionality) and CIS 2.3 (address unauthorized software) to prevent unapproved packages from reaching build environments. Implement D3-SFA (system file analysis) for build agent credential stores and D3-LAM (local account monitoring) for developer machine anomaly detection. Require MFA on all accounts with npm publish rights (CIS 6.5). Review GitHub Actions OIDC scope grants organization-wide and remove any that grant broader permissions than required (NIST AC-6, AC-5).

NIST Phase: Post Incident

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

Controls: NIST CM-7 (Least Functionality), NIST AC-5 (Separation of Duties), NIST AC-6 (Least Privilege), NIST AU-2 (Event Logging), NIST RA-3 (Risk Assessment), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 6.5 (Require MFA for Administrative Access), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For organizations without an enterprise proxy registry, implement a lightweight Verdaccio instance (free, open-source npm proxy) configured with an explicit allow-list of approved namespaces — block `@redhat-cloud-services` and any unreviewed namespace by default. Write a pre-commit hook using `npm audit` combined with a custom allow-list script that fails commits containing unapproved namespace additions to package.json. For OIDC scope auditing without enterprise GitHub tooling, run: `grep -r 'id-token: write'

.github/workflows/ --include='*.yml' -l` across your entire GitHub organization (clone all repos first) to enumerate every workflow with npm OIDC publish trust, then review each against the principle of least privilege. Deploy a Sysmon configuration with FileCreate events monitoring `~/.aws/credentials`, `~/.config/gcloud/`, and `~/.kube/config` on developer workstations to detect future Miasma-style credential file access.

Evidence: For the lessons-learned documentation, preserve the complete timeline of affected package download counts from npm's public download API (`https://api.npmjs.org/downloads/point/:/@redhat-cloud-services/`) for each of the 32 affected packages — this establishes your organization's exposure window and supports any regulatory notification decisions. Retain all cloud provider audit logs from the exposure window for a minimum of 12 months given the credential theft scope, as compromised cloud credentials may be used in delayed attacks well after the initial incident. Document the specific GitHub Actions workflow files and OIDC permission blocks that granted npm publish trust, as these serve as the root cause artifact for the control gap review.

Detection Guidance

Primary detection focus: identify whether any '@redhat-cloud-services' npm packages across 96 affected versions were installed in your environment, and whether post-install credential exfiltration occurred.

1. PACKAGE AUDIT: Search all repositories and build artifacts for 'package-lock.json' and 'yarn.lock' entries referencing '@redhat-cloud-services/*'. Use 'npm ls --all' or 'grep -r @redhat-cloud-services' across CI/CD workspace directories.

2. BUILD LOG REVIEW: Search GitHub Actions, Jenkins, GitLab CI, and CircleCI logs for 'npm install' or 'yarn' commands that resolved '@redhat-cloud-services' packages. Filter by date range from earliest known campaign activity to present.

3. NETWORK INDICATORS (T1567): On build agents and developer workstations, review outbound HTTP/HTTPS connections made during or immediately after npm install steps. Look for POST requests to unfamiliar or non-npm-registry external hosts originating from Node.js processes. Baseline comparison against known npm CDN infrastructure (registry.npmjs.org, 104.16.x.x ranges) will surface anomalies.

4. CREDENTIAL MISUSE: Query AWS CloudTrail for AssumeRole, GetSessionToken, or ListBuckets events from unexpected source IPs or user agents following build pipeline execution. In GCP, query Cloud Audit Logs for service account key usage from non-standard environments. In Azure, review Entra ID sign-in logs for service principal authentication anomalies.

5. SSH KEY ANOMALIES: Review SSH authentication logs on internal hosts for private key authentications from unexpected sources, particularly keys that reside on developer workstations or in CI/CD secret stores.

6. OIDC CONFIGURATION REVIEW: Audit all GitHub Actions OIDC trust relationships with npm, PyPI, and cloud providers. Identify any relationships where the 'subject' claim is broader than a single workflow or repository, these represent over-permissioned OIDC grants that expand blast radius under account compromise.

7. LOG CLEARING INDICATOR (T1070.003): On Linux build agents, check for truncated or missing ~/.bash_history and ~/.zsh_history files on accounts that executed npm install commands. Missing history following a pipeline run is a strong indicator of post-exploitation cleanup.

Controls supporting detection: NIST AU-2 (event logging), AU-6 (audit review and analysis), AU-12 (audit record generation), SI-4 (system monitoring); CIS 8.2 (collect audit logs).

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`@redhat-cloud-services npm namespace (32 packages, 96 versions)`	Entire npm namespace confirmed backdoored with Miasma malware; treat all packages and versions in this namespace as malicious until Red Hat confirms restored integrity	HIGH
URL	`https://www.npmjs.com/org/redhat-cloud-services`	npm organization page for the compromised namespace; verify package integrity and publication timestamps against Red Hat's official advisory before any reinstallation	HIGH

Framework Mappings

MITRE-ATTACK

- **T1554** — Compromise Host Software Binary
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1567** — Exfiltration Over Web Service
- **T1552.004** — Private Keys
- **T1552.001** — Credentials In Files
- **T1078.004** — Cloud Accounts
- **T1528** — Steal Application Access Token
- **T1059.007** — JavaScript
- **T1070.003** — Clear Command History
- **T1650** — Acquire Access
- **T1195.002** — Compromise Software Supply Chain

NIST-800-53R5

- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **IA-5** — Authenticator Management
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures

CIS-V8

- **5.2** — Use Unique Passwords
- **6.3** — Require MFA for Externally-Exposed Applications

- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **164.308(a)(5)(ii)(D)** — Password Management
- **164.312(d)** — Person or Entity Authentication

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

NIST-CSF-2

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

ISO-27001-2022

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1554	Compromise Host Software Binary	Persistence
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1567	Exfiltration Over Web Service	Exfiltration
T1552.004	Private Keys	Credential-Access
T1552.001	Credentials In Files	Credential-Access
T1078.004	Cloud Accounts	Defense-Evasion
T1528	Steal Application Access Token	Credential-Access
T1059.007	JavaScript	Execution
T1070.003	Clear Command History	Defense-Evasion
T1650	Acquire Access	Resource-Development
T1195.002	Compromise Software Supply Chain	Initial-Access

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/red-hat-npm-packages...	T3

Source	URL	Tier
npm Supply Chain Attack Compromises AntV - Orca Security	https://orca.security/resources/blog/antv-npm-supply-chain-attack/	T3
Typosquatted npm packages used to steal cloud and CI/CD secrets	https://www.microsoft.com/en-us/security/blog/2026/05/28/typosquatt...	T1
Large-Scale npm & PyPI Supply Chain Campaign Suspected Across ...	https://www.upwind.io/feed/large-scale-npm-pypi-supply-chain-campai...	T3
Shai-Hulud: Self-Replicating Worm Compromises 500+ NPM ...	https://www.stepsecurity.io/blog/ctrl-tinycolor-and-40-npm-packages...	T3

Generated 2026-06-02 06:31 UTC by TJS Security Command Center