INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-22 18:51 UTC

DPRK's Contagious Interview Campaign Adds Worm-Like Repository Propagation to Developer Targeting Playbook

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0204
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	Developer repositories and software pipelines; confirmed platforms include Next.js and Nx ecosystems; broader open-source repository platforms not fully confirmed in available sources
Published	2026-04-22T10:48:05
Discovery Source	Rss

Executive Summary

North Korea's Contagious Interview campaign has added worm-like propagation through poisoned developer repositories, including malicious versions of Next.js projects and Nx plugins, reducing the need for spear-phishing or direct operator contact to initiate infection. Any developer who clones or installs a compromised package can receive a remote access trojan, exposing their workstation, credentials, and connected systems. Organizations with software development pipelines face elevated supply chain risk: a single infected dependency can compromise build environments, source code, and downstream products.

Technical Analysis

The Contagious Interview campaign, attributed to Lazarus Group (DPRK), has expanded from spear-phishing job lures to supply chain compromise via malicious open-source packages. Poisoned Next.js repositories and malicious Nx plugin versions deliver remote access trojans (RATs) to developers who clone or install affected packages. Microsoft's security blog (February 24, 2026) documented C2 infrastructure tied to this developer-targeting activity. The attack reduces the social engineering prerequisite: passive repository interaction is sufficient for infection. Relevant CWEs: CWE-494 (Download of Code Without Integrity Check), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-506 (Embedded Malicious Code). MITRE ATT&CK techniques include T1195.001 and T1195.002 (Supply Chain Compromise), T1219 (Remote Access Software), T1071 (Application Layer Protocol), T1059/T1059.007 (Command and Scripting Interpreter),

T1080 (Taint Shared Content), T1204.002 (Malicious File execution), T1566.003 (Spearphishing via Service), and T1588.001 (Obtain Capabilities: Malware). No CVE has been assigned. Specific malicious package versions and associated file hashes have not been confirmed in available sources; check the Microsoft security blog and npm/registry advisories for updated IOC lists. Editorial assessment: source quality is moderate; treat specific package-version claims as requiring independent validation.

Action Checklist

1. Step 1: Containment. Audit all Next.js and Nx packages installed or cloned in the past 90 days. Cross-reference against your package-lock.json or yarn.lock files to identify any packages pulled from unofficial forks or unverified registry namespaces. Isolate developer workstations that cloned suspect repositories pending investigation.
2. Step 2: Detection. Review endpoint logs on developer workstations for unexpected outbound connections, especially to unfamiliar C2 domains or IPs. Query process execution logs for scripting interpreters (node, python, bash, powershell) spawned from package install hooks (npm postinstall, nx plugin lifecycle scripts). Monitor for credential access patterns following package installation events. Consult the Microsoft security blog (February 24, 2026) for domain, IP, and ASN indicators specific to this campaign's C2 infrastructure.
3. Step 3: Eradication. Remove any confirmed malicious packages and purge associated caches (npm cache clean, clear node_modules). Rotate credentials accessible from affected developer workstations, including tokens, SSH keys, cloud access keys, and repository secrets. Revoke and regenerate any CI/CD pipeline secrets on systems where compromised packages executed.
4. Step 4: Recovery. Rebuild affected developer environments from clean baselines rather than restoring from snapshots taken after potential infection. Verify build artifact integrity before promoting any code produced on affected systems to staging or production. Monitor for reinfection attempts via the same package namespaces for at least 30 days post-remediation.
5. Step 5: Post-Incident. Evaluate whether your organization enforces package integrity verification (npm audit signatures, lockfile pinning, private registry mirroring). Implement or strengthen controls aligned with CWE-494 and CWE-829: require signed packages, enforce allowlist-based registry policies, and block execution of postinstall scripts from unapproved packages. Map findings to NIST SP 800-161 supply chain risk management controls and MITRE ATT&CK T1195 mitigations.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to senior leadership, legal counsel, and relevant regulatory authorities if forensic analysis confirms the Contagious Interview RAT executed on any workstation with access to production credentials, customer PII, PHI, payment data, or code that was subsequently promoted to production — DPRK attribution and potential data exfiltration may trigger breach notification obligations under GDPR, HIPAA, or state breach notification laws; additionally escalate if CI/CD secrets were confirmed compromised, as downstream customer or partner systems may be at risk.

Recovery Notes	Rebuild all affected developer environments from pre-incident golden images; do not reuse any virtual machine snapshot, container image, or development environment provisioned after the earliest possible infection date within the 90-day audit window. Before returning any developer workstation to production use, verify that all CI/CD pipeline secrets, cloud IAM keys, SSH keys, and repository tokens have been rotated and that no artifacts produced on compromised systems have been promoted to staging or production without integrity verification. Maintain enhanced monitoring of npm registry activity for the specific package namespaces weaponized in this campaign — including typosquat variants — for a minimum of 30 days post-remediation, and retain network flow logs for at least 90 days to support retrospective analysis if new Contagious Interview infrastructure indicators are published by Microsoft or CISA.
Forensic Artifacts	npm postinstall execution chain logs: Sysmon Event ID 1 (Process Creation) entries where ParentImage matches node.exe or npm.exe and ChildImage matches cmd.exe, powershell.exe, bash, sh, or python — this is the primary execution path for RAT delivery via malicious Next.js or Nx postinstall lifecycle scripts in this campaign | Lockfile tampering evidence: diff of package-lock.json or yarn.lock against the project's git-committed version, specifically looking for resolved URLs pointing to GitHub forks or non-registry.npmjs.org hosts for next, nx, @nx/*, or related packages — the Contagious Interview worm propagates by substituting these entries | npm cache tarballs: contents of ~/.npm/_cacache or %APPDATA%/npm-cache containing the malicious package tarballs downloaded during the compromised install, preservable before npm cache clean and scannable with YARA rules for known Contagious Interview RAT strings or BeaverTail/InvisibleFerret malware family signatures associated with this DPRK campaign | Credential store access artifacts: filesystem access timestamps on ~/.aws/credentials, ~/.ssh/id_rsa, ~/.config/gh/hosts.yml, .env files, and browser-stored credential databases (Chrome Login Data, Firefox logins.json) showing access by node.exe or child processes spawned from npm lifecycle hooks during the suspected infection window | Outbound DNS and network connection logs: Sysmon Event ID 22 (DNS Query) and Event ID 3 (Network Connection) entries generated by node.exe or its child processes within 60 seconds of npm install or nx plugin execution, used to identify the initial C2 beacon domain and IP associated with the Contagious Interview campaign infrastructure

Per-Action IR Details

Step 1: Containment — Audit all Next.js and Nx packages installed or cloned in the past 90 days. Cross-reference against your package-lock.json or yarn.lock files to identify any packages pulled from unofficial forks or unverified registry namespaces. Isolate developer workstations that cloned suspect repositories pending investigation.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-8 (System Component Inventory) — implied; enumerate installed packages as system components, CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 1.2 (Address Unauthorized Assets)

Compensating: Run `npm ls --all --json > npm_tree_$(hostname).json` and `cat package-lock.json | jq '.packages | keys[]' | grep -v 'node_modules/@' | sort > pinned_deps.txt` on each developer workstation to dump the full resolved dependency tree. Compare resolved registry URLs in package-lock.json against the expected registry (e.g., registry.npmjs.org); any entry referencing a GitHub fork URL, a scoped namespace not belonging to your org, or an unfamiliar private registry host is a candidate for triage. Use `git log --all --oneline --graph` in cloned repos to check for anomalous commit authors or unsigned tags injected by the Contagious Interview operator. Network-isolate flagged workstations at the switch port or via OS firewall (`netsh advfirewall set allprofiles firewallpolicy

blockinbound,blockoutbound` on Windows; `ufw default deny outgoing && ufw default deny incoming` on Linux) before proceeding.

Evidence: Before isolating: capture a full memory image using WinPmem (Windows) or LiME kernel module (Linux) to preserve any in-memory RAT implant injected by the malicious Next.js or Nx postinstall hook. Snapshot the npm global cache directory (`%APPDATA%\npm-cache` on Windows; `~/.npm` on Linux/macOS) and the project `node_modules` directory verbatim — do not run `npm cache clean` yet. Preserve `package-lock.json` and `yarn.lock` with cryptographic hashes (`sha256sum package-lock.json`) before any remediation alters them. Document all resolved package versions and their registry source URLs as they exist at time of isolation, since the Contagious Interview worm propagates by substituting legitimate-looking package names in dependency trees.

Step 2: Detection — Review endpoint logs on developer workstations for unexpected outbound connections, especially to unfamiliar C2 domains or IPs. Query process execution logs for scripting interpreters (node, python, bash, powershell) spawned from package install hooks (npm postinstall, nx plugin lifecycle scripts). Monitor for credential access patterns following package installation events. Consult the Microsoft security blog (February 24, 2026) for C2 infrastructure indicators specific to this campaign.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Deploy Sysmon with a config tuned to log process creation (Event ID 1), network connections (Event ID 3), and DNS queries (Event ID 22) on developer workstations. Write a PowerShell query to identify Node.js spawning child shells: `Get-WinEvent -LogName 'Microsoft-Windows-Sysmon/Operational' | Where-Object {$_.Id -eq 1 -and $_.Message -match 'node.exe' -and $_.Message -match 'cmd.exe|powershell|bash|python'}`. On Linux/macOS, use `auditd` with a rule targeting `execve` calls where the parent process is `node` or `npm`: `auditctl -a always,exit -F arch=b64 -S execve -F ppid=$(pgrep -x npm) -k contagious_interview`. For network detection without a SIEM, run `ss -tunap` or `netstat -anb` immediately post-install and capture output; cross-reference outbound IPs against CISA and Microsoft-published Contagious Interview C2 indicators using a free tool like `grep -Ff c2_iocs.txt netstat_output.txt`. MITRE ATT&CK T1059.007 (JavaScript) and T1195.001 (Compromise Software Dependencies and Development Tools) are the primary technique references for this campaign's execution chain.

Evidence: Collect Sysmon Event ID 1 (Process Creation) logs filtering for `node.exe`, `npm.exe`, `npx.exe`, or `nx` as parent processes with child processes including `cmd.exe`, `powershell.exe`, `bash`, `sh`, or `python` — these represent postinstall script abuse specific to this campaign's RAT delivery mechanism. Capture Sysmon Event ID 3 (Network Connection) and Event ID 22 (DNS Query) logs generated within 60 seconds of any `npm install` or `nx` command execution to identify the initial C2 beacon. On Linux/macOS, collect `~/.bash_history`, `~/.zsh_history`, and shell session logs showing npm/yarn/nx invocations. Retrieve browser-stored credentials, SSH agent forwarding state (`ssh-add -l`), and any `.env` files, AWS credential files (`~/.aws/credentials`), and GitHub token stores (`~/.config/gh/hosts.yml`) accessible from the workstation at time of infection, since the Contagious Interview RAT specifically targets developer credential stores.

Step 3: Eradication — Remove any confirmed malicious packages and purge associated caches (npm cache clean, clear node_modules). Rotate credentials accessible from affected developer workstations, including tokens, SSH keys, cloud access keys, and repository secrets. Revoke and regenerate any CI/CD pipeline secrets on systems where compromised packages executed.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-3 (Malicious Code Protection), NIST AC-2 (Account Management) — credential rotation and revocation, CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 5.2 (Use Unique Passwords), CIS 6.2 (Establish an Access Revoking Process)

Compensating: After forensic preservation is complete, run `npm cache clean --force` and delete `node_modules` recursively (`rm -rf node_modules`). Revoke GitHub personal access tokens via GitHub Settings > Developer Settings

> Personal Access Tokens and force-expire all active sessions. Rotate AWS IAM keys using `aws iam delete-access-key` and `aws iam create-access-key` for any IAM user whose credentials were stored on the affected workstation. Revoke SSH keys by removing the compromised public key from all authorized_keys files on remote hosts (`grep -r 'COMPROMISED_KEY_FINGERPRINT' ~/.ssh/authorized_keys /etc/ssh/authorized_keys`). For CI/CD secrets (GitHub Actions, GitLab CI, Jenkins), rotate all secrets stored as environment variables or repository secrets and audit pipeline run logs for any jobs triggered by or on the compromised workstation during the exposure window. Use `git log --author` and pipeline audit logs to identify any commits or pipeline runs that may have exfiltrated secrets during the RAT's active dwell time.

Evidence: Before purging caches, copy the npm cache directory (`~/.npm/_cacache`) and collect file hashes of all installed package tarballs to enable offline malware analysis and YARA scanning for known Contagious Interview RAT components. Capture a list of all SSH keys, API tokens, and OAuth tokens present on the system at time of eradication (output of `ssh-add -l`, `cat ~/.ssh/id_*`, environment variable dumps) to establish the full scope of credential exposure. Document CI/CD pipeline secret names (not values) that were accessible to any process running under the compromised npm/nx lifecycle — check GitHub Actions secrets, `.env` files committed to the repo, and CI runner environment variables — since the Contagious Interview campaign specifically targets pipeline credentials to achieve downstream supply chain compromise.

Step 4: Recovery — Rebuild affected developer environments from clean baselines rather than restoring from snapshots taken after potential infection. Verify build artifact integrity before promoting any code produced on affected systems to staging or production. Monitor for reinfection attempts via the same package namespaces for at least 30 days post-remediation.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST SI-2 (Flaw Remediation), NIST CP-9 (System Backup) — referenced negatively; snapshot restoration is contraindicated here, NIST IR-4 (Incident Handling), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Provision fresh developer workstations or VM images from a known-good golden image predating the 90-day exposure window; do not restore from VM snapshots taken after the earliest possible infection date. Before reinstalling Next.js or Nx toolchains on rebuilt systems, pin all packages to verified versions using exact version specifiers in package.json (`"next": "14.2.5"` not `"^14.2.5"`) and validate package integrity using `npm audit signatures` (requires npm 8.x+) or verify SHA-512 hashes in lockfiles manually. For build artifact verification, generate SLSA provenance attestations or at minimum compute and store SHA-256 hashes of all build outputs (`find ./dist -type f -exec sha256sum {} \; > build_manifest.txt`) and compare against artifacts promoted before the incident. Monitor npm registry for new publications under the same namespaces used in the Contagious Interview campaign using `npm view time --json` to detect new malicious versions published post-remediation.

Evidence: Retain forensic copies of all build artifacts produced on affected systems during the exposure window — these must be analyzed before any code reaches staging or production, as the Contagious Interview RAT could have modified source files or injected backdoors into compiled outputs. Collect git diff outputs comparing pre-infection and post-infection commits on repositories cloned to affected workstations to identify any source code tampering. Preserve network flow logs or DNS query logs from the 30-day monitoring window post-remediation to detect reinfection attempts or ongoing C2 communication from previously undetected implants on systems not yet identified as compromised.

Step 5: Post-Incident — Evaluate whether your organization enforces package integrity verification (npm audit signatures, lockfile pinning, private registry mirroring). Implement or strengthen controls aligned with CWE-494 and CWE-829: require signed packages, enforce allowlist-based registry policies, and block execution of postinstall scripts from unapproved packages. Map findings to NIST SP 800-161 supply chain risk management controls and MITRE ATT&CK T1195 mitigations.

NIST Phase: Post Incident

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

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST SI-2 (Flaw Remediation), NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software)

Compensating: Configure npm to block postinstall script execution by default using `.npmrc` with `ignore-scripts=true`; document approved exceptions and re-enable selectively per package. Mirror approved Next.js and Nx packages to a private Verdaccio registry instance (free, self-hosted) and set `registry=http://your-verdaccio-host` in all developer `.npmrc` files to prevent direct pulls from npm public registry or GitHub-hosted forks. Write a Sigma rule targeting npm postinstall script abuse (parent: npm/node, child: scripting interpreter) and a YARA rule matching known Contagious Interview RAT string patterns for ongoing endpoint scanning with ClamAV. Document lessons learned in a formal post-incident report referencing NIST 800-61r3 §4 requirements, and share anonymized IOCs with sector ISAC partners per NIST IR-6 (Incident Reporting) obligations. Map control gaps to NIST SP 800-161r1 (C-SCRM) practices, specifically C-SCRM Level 2 controls around software bill of materials (SBOM) and supplier verification applicable to open-source dependency chains exploited by this campaign.

Evidence: Compile a final artifact package for lessons-learned review including: the original malicious package versions and their resolved registry sources from lockfiles, Sysmon/auditd logs showing the full postinstall execution chain, network logs showing C2 beacon timing relative to npm install events, and the complete list of credentials confirmed or suspected as exposed. Retain this package per NIST AU-11 (Audit Record Retention) for a minimum period consistent with your records retention policy, as DPRK-attributed campaigns have regulatory notification implications if developer workstations processed any customer PII or regulated data. Document the specific Next.js and Nx package namespaces exploited by Contagious Interview to update your private registry allowlist and prevent future ingestion of the same malicious namespace variants.

Detection Guidance

Primary behavioral indicators: scripting interpreters (node.js, python, bash, PowerShell) spawning network connections from package installation contexts; outbound connections to unfamiliar domains or IPs from developer workstations shortly after npm install, git clone, or nx plugin operations; unexpected scheduled tasks or persistence mechanisms created on developer endpoints. Log sources to query: EDR process creation and network connection telemetry on developer endpoints; DNS query logs for domains contacted following package installs; CI/CD pipeline execution logs for anomalous script execution during build phases. IOC note: specific C2 domains, IPs, and file hashes for this campaign are documented in Microsoft's February 24, 2026 security blog. Available sources did not surface confirmed IOC values for direct inclusion here; retrieve current indicators directly from that source before running threat hunts. For registry-level detection, monitor npm audit output and compare installed package checksums against official registry manifests.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://www.microsoft.com/en-us/security/blog/2026/02/24/c2-developer-targeting-campaign/`	Microsoft security blog documenting C2 infrastructure tied to malicious Next.js repository campaign — retrieve current IOC list from this source directly	HIGH

Framework Mappings

MITRE-ATTACK

- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1588.001** — Malware
- **T1195.002** — Compromise Software Supply Chain
- **T1219** — Remote Access Tools
- **T1071** — Application Layer Protocol
- **T1566.003** — Spearphishing via Service
- **T1059** — Command and Scripting Interpreter
- **T1059.007** — JavaScript
- **T1080** — Taint Shared Content
- **T1204.002** — Malicious File

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
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **SI-3** — Malicious Code Protection
- **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
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **15.1** — Establish and Maintain an Inventory of Service Providers

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

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1588.001	Malware	Resource-Development
T1195.002	Compromise Software Supply Chain	Initial-Access
T1219	Remote Access Tools	Command-And-Control
T1071	Application Layer Protocol	Command-And-Control
T1566.003	Spearphishing via Service	Initial-Access
T1059	Command and Scripting Interpreter	Execution
T1059.007	JavaScript	Execution
T1080	Taint Shared Content	Lateral-Movement
T1204.002	Malicious File	Execution

Sources

Source	URL	Tier
Security News	https://www.darkreading.com/cyberattacks-data-breaches/dprk-fake-jo...	T3
Developer-targeting campaign using malicious Next.js repositories	https://www.microsoft.com/en-us/security/blog/2026/02/24/c2-develop...	T1
Malicious versions of Nx and some supporting plugins were published	https://news.ycombinator.com/item?id=45034496	T3
Dev rejects CVE severity, makes his GitHub repo read-only - Facebook	https://www.facebook.com/groups/2600net/posts/3885382001684896/	T3
A Reality Check on SBOM-based Vulnerability Management - arXiv	https://arxiv.org/html/2511.20313v2	T2

DISCLAIMER

Generated 2026-04-22 18:51 UTC by TJS Security Command Center