INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-19 13:50 UTC

Nx Console VS Code Extension Supply Chain Compromise Enables Sigstore-Backed Downstream Package Poisoning

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0336
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Nx Console VS Code Extension (nrwl.angular-console v18.95.0); Microsoft Visual Studio Code; Cursor; JetBrains IDEs; 1Password; Anthropic Claude Code; npm; GitHub; AWS
Published	2026-05-19T03:49:23
Discovery Source	Rss

Executive Summary

On May 18, 2026, attackers compromised a developer's machine at Nrwl and pushed a malicious update to the Nx Console VS Code extension, exposing a large population of developers for approximately 11 minutes. The payload stole developer credentials across AWS, GitHub, npm, 1Password, and Anthropic Claude Code. The most serious business risk is downstream: stolen npm OIDC tokens were used to publish poisoned packages carrying valid cryptographic provenance signatures, meaning malicious code can enter your software supply chain appearing fully trusted and verified.

Technical Analysis

Affected component: Nx Console VS Code extension (nrwl.angular-console), version 18.95.0, distributed via the VS Code Marketplace. The attack originated from a compromised developer machine with publish access to the extension. The malicious payload is a multi-stage credential stealer targeting 1Password vaults, AWS credential files (~/.aws/credentials), GitHub tokens, npm authentication tokens (~/.npmrc), and Anthropic Claude Code credentials. Exfiltration uses multiple channels: HTTPS (T1071.001), GitHub API webhooks (T1102.001), and DNS tunneling (T1071.004). The novel escalation path: stolen npm OIDC tokens, when combined with Sigstore integration, allow the attacker to publish downstream npm packages with valid SLSA provenance attestations and cryptographic signatures (T1553.002). These packages pass signature verification in CI/CD pipelines that trust Sigstore-backed attestations, constituting supply chain compromise at the verification layer, not merely the package layer. Hardcoded credentials in exfiltration logic are not protected and are embedded within malicious

code. No CVE assigned at time of publication. Version 18.95.0 should be considered compromised; users should verify current installed version and rotate all credentials accessible from affected developer machines. Patch status: Nrwl pulled version 18.95.0 from the marketplace; users should confirm they are on a clean version. Sources: StepSecurity blog, The Hacker News, GitHub issue #1955 (all T3, search-retrieved, recommend human validation before actioning).

Action Checklist

1. Immediately identify all developer workstations where nrwl.angular-console version 18.95.0 was installed between May 18, 2026, and the time of remediation. Disable or remove the extension from VS Code, Cursor, and any JetBrains IDE that syncs VS Code extensions. Treat every identified machine as fully compromised for credential purposes.
2. Query VS Code extension telemetry and endpoint logs for installation of nrwl.angular-console v18.95.0. Search DNS logs for anomalous subdomain query patterns from developer workstations during the exposure window (DNS tunneling indicator). Review GitHub audit logs for unexpected token usage, npm publish events, and OAuth app authorizations. Check AWS CloudTrail for API calls using credentials stored in ~/.aws/credentials on affected machines. Look for HTTPS POST traffic to unknown external endpoints originating from IDE processes.
3. Rotate all credentials that were accessible on any affected developer machine: AWS IAM keys (revoke and reissue), GitHub personal access tokens and OAuth tokens, npm authentication tokens (~/.npmrc), 1Password emergency kit or service account credentials if the vault was unlocked during the exposure window, and Anthropic Claude Code API keys. Remove and reissue npm OIDC tokens used in CI/CD pipelines. Audit any npm packages published by affected developer accounts after May 18, 2026, for tampering, compare published package contents against source repository commits.
4. After credential rotation, audit all npm packages your organization published or consumed between May 18 and remediation date. Verify Sigstore/SLSA attestations on recently published packages against expected signing identities; a valid attestation signed by a compromised OIDC token will appear legitimate but must be treated as suspect. Re-run CI/CD pipelines against clean credentials and validate build outputs. Confirm affected extensions are uninstalled and replaced with a verified clean version from the VS Code Marketplace.
5. This attack exposed a structural gap: Sigstore-backed provenance attestations are only as trustworthy as the OIDC token issuance chain. Review your organization's trust policy for SLSA attestations and consider adding signer identity pinning (expected OIDC issuer and subject) rather than trusting any valid signature. Implement developer machine compromise detection. Enforce short-lived OIDC tokens for npm publish workflows. Evaluate whether developer IDE extensions are subject to the same software supply chain controls as production dependencies.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate immediately to CISO, legal counsel, and potentially SEC/breach notification counsel if any poisoned npm packages were published to the public registry or consumed by downstream customers, as this constitutes a supply chain breach with potential third-party harm and regulatory notification obligations under applicable state breach laws, GDPR Article 33, or SOC 2 contractual requirements.
Recovery Notes	After credential rotation and clean pipeline execution, monitor npm publish activity for all affected developer accounts and CI/CD service accounts for a minimum of 30 days, as threat actors who obtained OIDC tokens may have pre-positioned access in GitHub Actions environments not yet discovered. Treat any npm package published between May 18, 2026, and the completion of full credential rotation as potentially tampered — notify downstream consumers and consider yanking and republishing those versions with a verified clean build. Maintain heightened AWS CloudTrail alerting on the rotated IAM principals for at least 60 days to detect any delayed use of credentials exfiltrated before rotation completed.
Forensic Artifacts	VS Code extension host process artifacts: directory `~/.vscode/extensions/nrwl.angular-console-18.95.0/` including all bundled JavaScript files — the malicious payload would be embedded here as obfuscated JS executed in the extension host process; hash all files before removal | Sysmon Event ID 3 (Network Connection) records showing outbound HTTPS POST connections from `code --extensionHostProcess` (PID resolvable via Event ID 1 parent-child chain) to non-IDE infrastructure during the May 18 exposure window — these capture the exfiltration C2 beacon | AWS CloudTrail `GetCallerIdentity`, `ListBuckets`, and any `sts:AssumeRole` events sourced from developer workstation IPs using long-lived IAM keys from `~/.aws/credentials`, timestamped within the 11-minute exposure window or shortly after, indicating automated credential harvesting and immediate abuse | Sigstore Rekor transparency log entries (`rekor-cli search --email `) for npm packages published after May 18 — a poisoned package will show a valid OIDC attestation but the signing event timestamp and source IP will be anomalous relative to the developer's normal publish pattern | Shell history files (`~/.bash_history`, `~/.zsh_history`) and VS Code Electron SQLite databases at `~/.config/Code/Local Storage/leveldb/` which may contain cached GitHub OAuth tokens, npm session tokens, or Anthropic API keys written by the extension payload as part of its harvesting routine

Per-Action IR Details

Containment — Immediately identify all developer workstations where nrwl.angular-console version 18.95.0 was installed between May 18, 2026, and the time of remediation. Disable or remove the extension from VS Code, Cursor, and any JetBrains IDE that syncs VS Code extensions. Treat every identified machine as fully compromised for credential purposes.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), CIS 2.3 (Address Unauthorized Software), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Run `code --list-extensions | grep angular-console` on each developer workstation or deploy as a one-liner via Ansible/SSH: `ssh user@host 'code --list-extensions 2>/dev/null | grep -i angular-console'`. For JetBrains sync, check `~/.config/JetBrains/*/options/other.xml` or the JetBrains Toolbox sync directory for extension manifests. Force-remove with `code --uninstall-extension nrwl.angular-console`. On Windows, check `%USERPROFILE%\.vscode\extensions\nrwl.angular-console-18.95.0` and delete the directory. Document machine hostnames and timestamps before removal.

Evidence: Before removing the extension, preserve: the extension directory contents at `~/.vscode/extensions/nrwl.angular-console-18.95.0/` (hash all files with `sha256sum *` or `Get-FileHash`); VS Code extension installation logs at `~/.config/Code/logs/` (Linux) or `%APPDATA%\Code\logs\` (Windows); the extension's

`package.json` and any bundled JS files for malware analysis; Cursor extension path at `~/.cursor/extensions/`; JetBrains plugin cache at `~/.local/share/JetBrains/` or `%APPDATA%\JetBrains\`. Capture a full memory image if the IDE was running during the May 18 exposure window — the payload may have executed in the extension host process and left artifacts in heap memory.

Detection — Query VS Code extension telemetry and endpoint logs for installation of nrwl.angular-console v18.95.0. Search DNS logs for anomalous subdomain query patterns from developer workstations during the exposure window (DNS tunneling indicator). Review GitHub audit logs for unexpected token usage, npm publish events, and OAuth app authorizations. Check AWS CloudTrail for API calls using credentials stored in ~/.aws/credentials on affected machines. Look for HTTPS POST traffic to unknown external endpoints originating from IDE processes.

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), CIS 8.2 (Collect Audit Logs)

Compensating: DNS tunneling detection: export DNS query logs from Pi-hole, BIND query logs, or router syslog and run `awk '{print $NF}' dns.log | awk -F. '{print $(NF-1)"."$NF}' | sort | uniq -c | sort -rn` — look for high-entropy subdomains or unusually high query counts to a single second-level domain from IDE processes. For GitHub audit logs, use `gh api /orgs/{org}/audit-log --paginate | jq '.[] | select(.created_at >= "2026-05-18T00:00:00Z")' > github_audit.json` and filter on `action: "token.create"`, `npm.publish`, and `oauth_application.create`. For AWS CloudTrail without SIEM, use AWS CLI: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue= --start-time 2026-05-18T00:00:00Z | jq '.Events[] | {EventTime, EventName, SourceIPAddress}'`. For process-level network traffic, deploy Sysmon with Event ID 3 (Network Connection) filtering on `code.exe`, `cursor.exe`, and `idea64.exe` as parent processes making outbound connections to non-Microsoft, non-GitHub IP ranges.

Evidence: Collect before analysis: Sysmon Event ID 1 (Process Create) and Event ID 3 (Network Connection) logs for `code --extensionHostProcess` and child processes spawned during May 18 exposure window; DNS resolver cache snapshots (`ipconfig /displaydns` on Windows, `resolvectl query` on Linux) from affected workstations captured before reboot; AWS CloudTrail logs filtered to the IAM principal associated with `~/.aws/credentials` on each affected machine for the window 2026-05-18T00:00:00Z to remediation time; GitHub audit log exports showing token last-used timestamps and source IPs; `~/.npmrc` file (contains npm auth token — do not transmit in plaintext, hash and vault); browser localStorage and session storage from VS Code's Electron renderer process at `~/.config/Code/Local Storage/` which may contain cached OAuth tokens.

Eradication — Rotate all credentials that were accessible on any affected developer machine: AWS IAM keys (revoke and reissue), GitHub personal access tokens and OAuth tokens, npm authentication tokens (~/.npmrc), 1Password emergency kit or service account credentials if the vault was unlocked during the exposure window, and Anthropic Claude Code API keys. Remove and reissue npm OIDC tokens used in CI/CD pipelines. Audit any npm packages published by affected developer accounts after May 18, 2026, for tampering — compare published package contents against source repository commits.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST IA-5 (Authenticator Management), NIST AC-2 (Account Management), NIST SI-2 (Flaw Remediation), CIS 5.2 (Use Unique Passwords), CIS 6.2 (Establish an Access Revoking Process)

Compensating: AWS IAM rotation: `aws iam list-access-keys --user-name ` to enumerate, then `aws iam update-access-key --access-key-id --status Inactive` to disable before deletion. GitHub PAT revocation: navigate to Settings > Developer settings > Personal access tokens and revoke all tokens; for org-level, use `gh api -X DELETE /user/installations/` for OAuth apps. npm token rotation: `npm token revoke ` for each token listed via `npm token list`. For CI/CD OIDC token reissuance, update the GitHub Actions OIDC trust policy in `~/.github/workflows/*.yml` to require a new subject claim format and rotate the npm automation token in repo secrets. Package tampering check: for each post-May-18 published package, run `npm pack @ --dry-run` locally from a clean machine and diff against `git

archive HEAD | tar -tz` output — any file present in the published tarball but absent from the git tree is a red flag.

Evidence: Before rotating credentials, preserve forensic copies of: `~/.aws/credentials` and `~/.aws/config` (vault these — do not email); `~/.npmrc` full contents including registry auth tokens; VS Code Electron keychain entries via `secret-tool search --all application 'VS Code'` (Linux) or Windows Credential Manager export (`cmdkey /list`); 1Password CLI session token cache at `~/.op/` if the agent was running; Anthropic API key references in shell history (`~/.bash_history`, `~/.zsh_history`) and in `.env` files recursively under the developer's home directory (`find ~ -name '.env' -exec grep -l ANTHROPIC {} \;`); GitHub Actions workflow run logs for any pipeline executions triggered by affected accounts on or after May 18, 2026.

Recovery — After credential rotation, audit all npm packages your organization published or consumed between May 18 and remediation date. Verify Sigstore/SLSA attestations on recently published packages against expected signing identities — a valid attestation signed by a compromised OIDC token will appear legitimate but must be treated as suspect. Re-run CI/CD pipelines against clean credentials and validate build outputs. Confirm affected extensions are uninstalled and replaced with a verified clean version from the VS Code Marketplace.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST IR-4 (Incident Handling), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Sigstore attestation verification: use the `cosign` CLI (`cosign verify-attestation --type slsaprovenance `) and cross-check the `Issuer` and `Subject` fields in the Rekor transparency log entry against your known-good OIDC subject (e.g., `https://github.com///.github/workflows/publish.yml@refs/heads/main`). Any attestation where the OIDC subject matches an affected developer's identity or an unexpected workflow path should be treated as poisoned. For package content auditing without tooling: `npm pack @` to download the tarball, then `tar -tzf | sha256sum` and compare against a pre-incident hash from your artifact registry or `npm view @ dist.shasum`. Clean VS Code extension reinstall: verify the Marketplace version hash via `code --install-extension nrwl.angular-console@` only after confirming the clean version hash against the publisher's GitHub release SHA.

Evidence: Collect before clearing systems for reuse: Sigstore Rekor transparency log entries for all npm packages published by affected accounts between May 18 and remediation — retrieve via `rekor-cli search --email --format json > rekor_entries.json`; npm publish audit trail from `npm audit` and registry publish history (`npm info time`); CI/CD pipeline artifact hashes from GitHub Actions run summaries for any workflow triggered during the exposure window; a before/after diff of `package-lock.json` and `yarn.lock` files in any repository where affected developers merged PRs after May 18 — dependency confusion or subtle version pin changes are a secondary attack vector.

Post-Incident — This attack exposed a structural gap: Sigstore-backed provenance attestations are only as trustworthy as the OIDC token issuance chain. Review your organization's trust policy for SLSA attestations and consider adding signer identity pinning (expected OIDC issuer and subject) rather than trusting any valid signature. Implement developer machine compromise detection. Enforce short-lived OIDC tokens for npm publish workflows. Evaluate whether developer IDE extensions are subject to the same software supply chain controls as production dependencies.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SA-12 (Supply Chain Protection), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-7 (Least Functionality), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: OIDC signer identity pinning: in your GitHub Actions workflow, add a `permissions: id-token: write` scope restriction and use `cosign` with `--certificate-identity-regexp` and `--certificate-oidc-issuer` flags to enforce that only your org's specific workflow subject can be considered a valid publisher. IDE extension allowlisting: create a `.vscode/extensions.json` with a controlled `recommendations` list, enforce via a pre-commit hook that fails if `code

--list-extensions` output contains extensions not on the allowlist (`comm -23 <(code --list-extensions | sort) <(sort allowlist.txt)`). Developer machine compromise detection without EDR: deploy Sysmon with a community config (e.g., SwiftOnSecurity or Olaf Hartong's modular config) and forward Event IDs 1, 3, 7, 11, 22 to a central syslog server; write a Sigma rule targeting `code --extensionHostProcess` spawning `cmd.exe`, `powershell.exe`, or `curl` as child processes — these are anomalous for a legitimate IDE extension.

Evidence: For the lessons-learned report, preserve and analyze: the full Rekor transparency log search results for all packages published via compromised OIDC tokens to establish the blast radius of downstream poisoning; a timeline reconstruction built from GitHub audit log, AWS CloudTrail, and npm publish history showing the sequence from extension installation to credential exfiltration to downstream package publish; the malicious extension's bundled JavaScript (deobfuscated if necessary using `node --inspect` or `js-beautify`) to document exact exfiltration targets and methods for detection rule development; any network PCAP captured from affected workstations during the exposure window showing the C2 POST requests, for IOC extraction (destination IPs, URI patterns, User-Agent strings, payload structure).

Detection Guidance

1. Extension inventory: Query endpoint management (Intune, Jamf, osquery) for installed VS Code extension nrwl.angular-console at version 18.95.0 across all developer machines. 2. DNS anomalies: Search DNS resolver logs for high-frequency subdomain queries from developer workstations during May 18, 2026, especially patterns consistent with data encoding in subdomains (long random-looking labels, high query rate to a single apex domain). 3. GitHub audit log: Filter for token creation, OAuth authorization, and unexpected repository access events on May 18 from accounts on affected machines. 4. npm publish audit: Query npm registry for packages published by developer accounts associated with affected machines after May 18, cross-reference against expected release schedules. 5. AWS CloudTrail: Search for API calls from access keys stored in ~/.aws/credentials on flagged machines, particularly calls outside normal working hours or from unexpected source IPs. 6. Process behavior: On endpoints with EDR, look for IDE processes (code, cursor, idea) spawning child processes making outbound HTTPS connections to non-Microsoft, non-JetBrains endpoints. 7. Sigstore attestation audit: For any npm packages published post-May 18 by affected accounts, verify the OIDC subject and issuer in the Sigstore transparency log (search Rekor) matches expected CI/CD identity, not a developer workstation identity.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`Not publicly disclosed at time of publication`	File hash for malicious nrwl.angular-console v18.95.0 VSIX package — check StepSecurity blog and GitHub issue #1955 for updated indicators	LOW
URL	`Not publicly disclosed at time of publication`	HTTPS exfiltration endpoint used by the payload — monitor StepSecurity disclosure for updated network IOCs	LOW

Type	Value	Context	Confidence
DOMAIN	`Not publicly disclosed at time of publication`	DNS tunneling apex domain used for credential exfiltration — monitor StepSecurity disclosure for updated DNS IOCs	LOW

Framework Mappings

MITRE-ATTACK

- **T1553.002** — Code Signing
- **T1543.001** — Launch Agent
- **T1071.001** — Web Protocols
- **T1027** — Obfuscated Files or Information
- **T1071.004** — DNS
- **T1554** — Compromise Host Software Binary
- **T1528** — Steal Application Access Token
- **T1102.001** — Dead Drop Resolver
- **T1059.006** — Python
- **T1036.001** — Invalid Code Signature
- **T1552.001** — Credentials In Files
- **T1555** — Credentials from Password Stores
- **T1547.011**
- **T1059.007** — JavaScript
- **T1567.001** — Exfiltration to Code Repository
- **T1195.001** — Compromise Software Dependencies and Development Tools

NIST-800-53R5

- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **IA-5** — Authenticator Management
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

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

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **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

ISO-27001-2022

- **A.8.28** — Secure coding
- **A.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain
- **A.8.24** — Use of cryptography
- **A.5.23** — Information security for use of cloud services

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1553.002	Code Signing	Defense-Evasion
T1543.001	Launch Agent	Persistence
T1071.001	Web Protocols	Command-And-Control
T1027	Obfuscated Files or Information	Defense-Evasion
T1071.004	DNS	Command-And-Control
T1554	Compromise Host Software Binary	Persistence
T1528	Steal Application Access Token	Credential-Access
T1102.001	Dead Drop Resolver	Command-And-Control
T1059.006	Python	Execution

Technique ID	Technique Name	Tactic
T1036.001	Invalid Code Signature	Defense-Evasion
T1552.001	Credentials In Files	Credential-Access
T1555	Credentials from Password Stores	Credential-Access
T1547.011		
T1059.007	JavaScript	Execution
T1567.001	Exfiltration to Code Repository	Exfiltration
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/compromised-nx-console-18950-targ...	T3
Nx Console VS Code Extension Compromised - StepSecurity	https://www.stepsecurity.io/blog/nx-console-vs-code-extension-compr...	T3
Nx Console unable to start in VS Code #1955 - GitHub	https://github.com/nrwl/nx-console/issues/1955	T3
nrwl/nx + VS Code: Nx Console won't connect with project	https://stackoverflow.com/questions/70413395/nrwl-nx-vs-code-nx-con...	T3
Nx Console - Visual Studio Marketplace	https://marketplace.visualstudio.com/itemdetails?itemName=nrwl.angu...	T3

Generated 2026-05-19 13:50 UTC by TJS Security Command Center