INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-16 07:11 UTC

North Korean Threat Clusters Industrialize Developer Supply Chain Operations Across VS Code, npm, and GitHub

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0474
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Microsoft Visual Studio Code, VS Code Marketplace (VSIX extensions), npm registry, GitHub, Packagist, Cursor IDE, Microsoft SharePoint (Graph API abused for C2), macOS, Linux, Windows, Telegram, browser-based cryptocurrency wallet extensions
Published	2026-06-15T15:32:52
Discovery Source	Rss

Executive Summary

Multiple North Korea-aligned threat groups have systematically embedded malware into developer tools, including VS Code extensions, npm packages, and GitHub repositories, to steal cryptocurrency and credentials at industrial scale. Organizations whose developers use these ecosystems face reported financial losses exceeding $12 million and compromise of 26,584 cryptocurrency wallets in Q1 2026 alone, with a single Cursor IDE extension incident resulting in $500,000 in direct losses (per early 2026 threat intelligence reporting). The operation's ability to reconstitute infrastructure after marketplace takedowns signals a sustained, well-resourced campaign with no near-term end in sight.

Technical Analysis

Contagious Interview (aka Famous Chollima) and a newly identified cluster designated UNK_DeadDrop have industrialized developer supply chain compromise across VS Code Marketplace (VSIX), npm, Packagist, GitHub, and Cursor IDE. Attack vectors include malicious IDE extensions distributed through official and third-party marketplaces, trojanized npm packages with embedded Git hooks (CWE-506, CWE-829), and weaponized GitHub repositories used in social engineering lures tied to fake job interviews and coding challenges. Malware capabilities include credential harvesting from password stores (T1555), session cookie theft (T1539), and browser extension targeting of cryptocurrency wallets (T1176). C2 channels abuse Microsoft SharePoint via the Graph API (T1102.002) and Telegram to blend with legitimate enterprise traffic, complicating

network-layer detection. Persistence is achieved via boot/logon autostart mechanisms (T1547). Payloads are delivered cross-platform targeting macOS, Linux, and Windows, using JavaScript (T1059.007), Unix shell (T1059.004), and Visual Basic (T1059.005) interpreters. Obfuscation is consistent across samples (T1027). No CVE is associated with this campaign; exploitation depends entirely on user execution (T1204.002) and supply chain trust abuse (T1195.001). Source quality score is 0.64, reflecting T3 reporting; organizations should validate IOCs against internal telemetry before acting on specifics.

Action Checklist

1. Step 1: Containment, Immediately audit all VS Code, Cursor IDE, and npm packages installed on developer workstations. Restrict or block installation of marketplace extensions from unverified publishers organization-wide. Enforce allowlists for approved extensions via policy before developers can install new tooling. Reference NIST AC-3 (Access Enforcement) and CIS 2.3 (Address Unauthorized Software).
2. Step 2: Detection, Query endpoint telemetry for outbound connections to SharePoint Graph API endpoints (graph.microsoft.com) originating from IDE processes or Node.js/npm runtime outside expected business applications; these are anomalous C2 indicators. Hunt for Git hook scripts (pre-commit, post-merge) in developer repositories containing encoded or obfuscated payloads. Review browser extension inventories on developer machines for unrecognized cryptocurrency wallet extensions (T1176). Enable and review AU-2 (Event Logging) for process execution events from IDE plugin directories. Cross-reference installed npm package manifests against known-malicious package lists published by Snyk and Socket.dev.
3. Step 3: Eradication, Remove any unrecognized or publisher-unverified IDE extensions and npm packages from affected systems. Purge and regenerate npm lockfiles on projects where trojanized dependencies may have been introduced. Rotate all credentials and API tokens accessible from compromised developer machines, prioritizing cloud provider keys, code signing certificates, and CI/CD pipeline secrets. Apply CIS 5.3 (Disable Dormant Accounts) and CIS 5.4 (Restrict Administrator Privileges) to developer accounts showing anomalous activity. Reference D3-CRO (Credential Rotation) and D3-CH (Credential Hardening).
4. Step 4: Recovery, Validate clean extension inventories against an approved publisher allowlist before returning developer workstations to production pipelines. Monitor for resumed outbound Graph API anomalies for a minimum of 30 days post-remediation. Confirm no unauthorized commits or modified Git hooks exist in internal repositories. Audit CI/CD pipeline integrity; verify no injected steps were introduced during the compromise window. Apply D3-SFA (System File Analysis) to confirm IDE configuration files and startup scripts are unmodified per NIST SI-4 (System Monitoring) equivalents.
5. Step 5: Post-Incident, This campaign directly exposes gaps in software supply chain governance. Formalize an approved extension and package registry policy enforced via tooling (CIS 2.1, CIS 2.3). Implement dependency integrity verification for all npm packages (mitigating CWE-494, Download of Code Without Integrity Check). Establish ongoing monitoring of open-source information channels for newly identified malicious packages per NIST AU-13 (Monitoring for Information Disclosure). Brief developers on fake job interview lures as a primary initial access vector.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to executive leadership, legal counsel, and your incident response retainer immediately if any of the following are confirmed: (1) code signing certificates accessible from compromised developer machines are unaccounted for, indicating potential software supply chain poisoning of your own published packages; (2) CI/CD pipeline secrets were exposed, indicating downstream customer or partner environments may be affected; (3) cryptocurrency wallet keys or financial API credentials were accessible, triggering direct financial loss notification obligations; (4) PII or regulated data (PHI, PCI-scoped cardholder data) was accessible from the compromised developer environment, triggering breach notification requirements under applicable regulations (GDPR 72-hour window, US state breach laws).
Recovery Notes	Do not return any developer workstation to production pipelines until a clean extension and npm dependency inventory has been validated against an approved allowlist and all CI/CD secrets have been rotated and confirmed revoked in the originating identity provider. Monitor all internal repositories for unauthorized commits and all CI/CD pipeline executions for anomalous external callouts for a minimum of 30 days post-remediation, as this campaign has demonstrated persistent re-entry via poisoned dependencies re-introduced through developer cloning activity. Treat any resumed outbound connection from an IDE or Node.js process to graph.microsoft.com as evidence of incomplete eradication requiring full workstation reimaging rather than incremental cleanup.
Forensic Artifacts	VS Code and Cursor IDE extension directories (~/.vscode/extensions/ on macOS/Linux, %USERPROFILE%\.vscode\extensions\ on Windows): each installed extension's extracted VSIX folder contains the malicious JavaScript payload delivered by North Korean threat actors in this campaign; preserve with directory timestamps and SHA-256 hashes before any removal | Git hook files (.git/hooks/pre-commit, post-merge, post-checkout) across all developer repository clones: this campaign specifically plants Base64-encoded or obfuscated shell/Python payloads in these hooks to achieve persistent code execution on developer workstations at commit and merge events | npm postinstall script execution logs (~/.npm/_logs/ and node_modules/.package-lock.json): North Korean-attributed packages in this campaign deliver malware via postinstall hooks; these logs record script execution timestamps and can reconstruct which package version triggered the payload | Network connection records showing IDE or Node.js processes communicating with graph.microsoft.com: this campaign uniquely abuses the Microsoft SharePoint Graph API as a C2 channel, making graph.microsoft.com connections from developer tooling the single most reliable forensic indicator of active compromise; capture via Sysmon Event ID 3 or pcap before any network isolation | Browser extension storage for cryptocurrency wallet extensions (Chrome: %LOCALAPPDATA%\Google\Chrome\User Data\Default\Extensions\; macOS: ~/Library/Application Support/Google/Chrome/Default/Extensions/): this campaign injected malicious browser extensions to intercept and steal cryptocurrency wallet credentials and session tokens; the extension storage SQLite databases contain harvested credential fragments recoverable even after the extension is removed

Per-Action IR Details

Step 1: Containment — Immediately audit all VS Code, Cursor IDE, and npm packages installed on developer workstations. Restrict or block installation of marketplace extensions from unverified publishers organization-wide. Enforce allowlists for approved extensions via policy before developers can install new tooling. Reference NIST AC-3 (Access Enforcement) and CIS 2.3 (Address Unauthorized Software).

NIST Phase: Containment

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

Controls: NIST AC-3 (Access Enforcement), NIST AC-6 (Least Privilege), CIS 2.3 (Address Unauthorized Software), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: On each developer workstation, run `code --list-extensions > extensions_$(hostname).txt` for VS Code and the equivalent `cursor --list-extensions` for Cursor IDE to generate a baseline inventory. Cross-reference output against your approved publisher allowlist using a simple diff script. Block outbound VSIX download domains at the perimeter firewall (marketplace.visualstudio.com, open-vsx.org) for unapproved hosts. For npm, run `npm ls --depth=0 --json > npm_inventory_$(hostname).json` in each project root and diff against a known-good manifest. These audits must complete before any credential rotation to preserve the pre-remediation software state as forensic evidence.

Evidence: Before blocking or uninstalling any extension, capture: (1) the full installed extension list with version and publisher metadata (`code --list-extensions --show-versions`); (2) the VS Code and Cursor IDE extension directories (`~/.vscode/extensions/` on macOS/Linux, `%USERPROFILE%\.vscode\extensions\` on Windows) — preserve directory timestamps and file hashes (SHA-256) of all VSIX-extracted folders; (3) active network connections from IDE processes using `Get-NetTCPConnection -OwningProcess (Get-Process code).Id` (Windows) or `lsof -i -p $(pgrep -d, code)` (macOS/Linux) to capture live C2 beacons to graph.microsoft.com before network controls sever them; (4) running process tree showing IDE parent-child relationships (`Get-CimInstance Win32_Process | Select ProcessId,ParentProcessId,Name,CommandLine` or `ps auxf`).

Step 2: Detection — Query endpoint telemetry for outbound connections to SharePoint Graph API endpoints (graph.microsoft.com) originating from IDE processes or Node.js/npm runtime outside expected business applications; these are anomalous C2 indicators. Hunt for Git hook scripts (pre-commit, post-merge) in developer repositories containing encoded or obfuscated payloads. Review browser extension inventories on developer machines for unrecognized cryptocurrency wallet extensions (T1176). Enable and review AU-2 (Event Logging) for process execution events from IDE plugin directories. Cross-reference installed npm package manifests against known-malicious package lists published by Snyk and Socket.dev.

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 AU-12 (Audit Record Generation), CIS 8.2 (Collect Audit Logs)

Compensating: Deploy Sysmon with a configuration that logs Event ID 3 (Network Connection) and Event ID 1 (Process Creation): filter for `code.exe`, `cursor.exe`, `node.exe`, or `npm.exe` initiating connections to `graph.microsoft.com` — any such connection outside sanctioned Microsoft 365 business apps is a confirmed C2 indicator for this campaign. Use the following PowerShell hunt: `Get-WinEvent -LogName 'Microsoft-Windows-Sysmon/Operational' | Where-Object {$_.Id -eq 3 -and $_.Message -match 'graph.microsoft.com' -and $_.Message -match '(code|cursor|node|npm)'}`. For Git hook hunting, run `find /path/to/repos -name 'pre-commit' -o -name 'post-merge' -o -name 'post-checkout' | xargs grep -lE '(base64|eval|exec|curl|wget|python -c)' 2>/dev/null` across all local repository clones. For browser extension enumeration on Chrome/Edge, parse `%LOCALAPPDATA%\Google\Chrome\User Data\Default\Extensions\` or equivalent and cross-reference extension IDs against known-malicious wallet extension lists from Socket.dev threat reports.

Evidence: Capture before any remediation action alters live state: (1) memory image of any running IDE or Node.js process showing anomalous outbound connections (use WinPmem on Windows or `osxpmem` on macOS) — in-memory payloads delivered via this campaign's VSIX extensions will not survive process termination; (2) full output of `netstat -ano` (Windows) or `ss -tulpn` (Linux/macOS) cross-referenced against process IDs for IDE and Node.js runtimes; (3) contents of all Git hook files (`.git/hooks/pre-commit`, `post-merge`, `post-checkout`) in developer repository clones — this campaign specifically plants obfuscated payloads here; (4) browser extension storage directories for any cryptocurrency wallet extensions to recover injected scripts before browser process termination; (5) npm package `postinstall` script contents from `node_modules/.package-lock.json` and individual `package.json` files — North Korean supply chain packages in this campaign execute payloads via postinstall hooks.

Step 3: Eradication — Remove any unrecognized or publisher-unverified IDE extensions and npm packages from affected systems. Purge and regenerate npm lockfiles on projects where trojanized dependencies may have been introduced. Rotate all credentials and API tokens accessible from compromised developer machines, prioritizing cloud provider keys, code signing certificates, and CI/CD pipeline secrets. Apply CIS

5.3 (Disable Dormant Accounts) and CIS 5.4 (Restrict Administrator Privileges) to developer accounts showing anomalous activity. Reference D3-CRO (Credential Rotation) and D3-CH (Credential Hardening).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), CIS 5.3 (Disable Dormant Accounts), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: For extension removal: `code --uninstall-extension ` for each flagged VSIX. For npm purge: `rm -rf node_modules package-lock.json && npm cache clean --force && npm install` using a pinned, verified `package.json` restored from version control before the compromise window. For credential rotation without a PAM tool: enumerate exposed secrets by scanning developer home directories and project roots with `truffleHog --regex --entropy=False .` or `gitleaks detect --source . --verbose` against local repo history to identify any secrets that were already committed and are now in threat actor hands before rotating. Revoke and reissue AWS IAM keys, GCP service account JSON keys, GitHub PATs, npm publish tokens, and code signing certificates — in that order of blast radius. Document each rotation with timestamp for the post-incident timeline.

Evidence: CRITICAL — all volatile evidence from Steps 1 and 2 must be fully captured before this step. Additionally capture before credential rotation: (1) environment variable dumps from all running IDE and shell processes (`Get-ChildItem Env:` on Windows; `cat /proc//environ` on Linux) — this campaign's malware harvests credentials from environment variables accessible to developer tooling; (2) contents of `~/.aws/credentials`, `~/.config/gcloud/`, `~/.npmrc`, `~/.gitconfig`, and any `.env` files in project directories — these are confirmed exfiltration targets for North Korean developer-focused operations; (3) CI/CD pipeline configuration files (`.github/workflows/*.yml`, `.gitlab-ci.yml`, `Jenkinsfile`) for injected malicious steps introduced during the compromise window, preserved with file hashes before any modification.

Step 4: Recovery — Validate clean extension inventories against an approved publisher allowlist before returning developer workstations to production pipelines. Monitor for resumed outbound Graph API anomalies for a minimum of 30 days post-remediation. Confirm no unauthorized commits or modified Git hooks exist in internal repositories. Audit CI/CD pipeline integrity; verify no injected steps were introduced during the compromise window. Apply D3-SFA (System File Analysis) to confirm IDE configuration files and startup scripts are unmodified per NIST SI-4 (System Monitoring) equivalents.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AC-17 (Remote Access), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Before returning any workstation to production pipelines, generate a fresh extension inventory (`code --list-extensions --show-versions`) and compare SHA-256 hashes of installed VSIX package contents against publisher-signed hashes where available. For Git repository integrity, run `git log --all --oneline --graph` across all internal repos for the compromise window timeframe and `git diff HEAD~N -- .git/hooks/` to detect hook modifications. For CI/CD pipeline audit without enterprise tooling, parse workflow YAML diffs using `git log -p -- .github/workflows/` to identify any injected steps that call external URLs or execute encoded payloads. Maintain a Sysmon-based watch for 30 days specifically filtering for `graph.microsoft.com` connections from IDE or Node.js processes — a resumed beacon indicates persistence was not fully eradicated (e.g., a missed Git hook or a re-cloned poisoned dependency).

Evidence: No new volatile capture required at this phase if Steps 1–3 were executed correctly. However, before signing off on workstation return to production, preserve: (1) a post-remediation baseline snapshot of the extension directory with file hashes for audit chain-of-custody; (2) a Git commit log export covering the entire compromise window for every internal repository the affected developer had write access to — unauthorized commits by this threat actor may include subtle backdoors in application code that survive the tool-layer eradication; (3) a final `netstat`/`ss` sweep confirming no residual connections to graph.microsoft.com, Telegram API endpoints, or other C2 infrastructure identified in threat intelligence for this campaign.

Step 5: Post-Incident — This campaign directly exposes gaps in software supply chain governance. Formalize an approved extension and package registry policy enforced via tooling (CIS 2.1, CIS 2.3). Implement dependency integrity verification for all npm packages (mitigating CWE-494, Download of Code Without Integrity Check). Establish ongoing monitoring of open-source information channels for newly identified malicious packages per NIST AU-13 (Monitoring for Information Disclosure). Brief developers on fake job interview lures as a primary initial access vector.

NIST Phase: Post Incident

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

Controls: NIST AU-13 (Monitoring for Information Disclosure), NIST AU-2 (Event Logging), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.3 (Address Unauthorized Software), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Operationalize supply chain monitoring for a 2-person team: subscribe to Socket.dev and Snyk's OSS threat feeds (both offer free tiers) and configure RSS or webhook alerts for newly flagged npm packages matching your dependency graph. Integrate `npm audit` and `pip-audit` into pre-commit hooks organization-wide using a Husky (npm) hook enforced in a shared `.husky/` directory committed to a template repository. Enforce npm `--ignore-scripts` flag in CI/CD pipelines to block postinstall payload execution — the primary delivery mechanism for North Korean supply chain packages in this campaign. For developer awareness, incorporate a specific scenario module into security training: North Korean threat actors in this campaign used LinkedIn and Telegram fake recruiter personas to socially engineer developers into cloning and running malicious coding challenge repositories, a vector that bypasses all technical controls if developers are unaware.

Evidence: Post-incident documentation to preserve for lessons learned and potential regulatory notification: (1) timeline of the compromise window derived from Git commit history, extension install timestamps, and npm package install dates from `~/.npm/_logs/`; (2) full list of credentials and tokens confirmed exposed, with rotation confirmation timestamps, for audit trail; (3) any npm packages or VSIX extensions identified as malicious during the incident, with SHA-256 hashes, for sharing with CISA, Snyk, and Socket.dev to protect the broader developer community; (4) developer interview notes documenting whether any team members received unsolicited job offers, coding challenges, or repository links via LinkedIn or Telegram in the 60 days preceding detection — this reconstructs the initial access vector for the lessons-learned report and threat intelligence sharing.

Detection Guidance

Primary detection focus: anomalous outbound traffic from IDE or Node.js processes to Microsoft Graph API (graph.microsoft.com), especially from developer endpoints with no sanctioned SharePoint integration. Hunt for npm packages with lifecycle scripts (install, prepare, postinstall) containing base64-encoded or obfuscated payloads; tools such as Socket.dev and Snyk's CLI can assist. Inspect Git hook files (located in .git/hooks/) across local and CI/CD repositories for unexpected shell, JavaScript, or VBScript content. On macOS and Linux, review launchd plist files and cron entries created by IDE processes for persistence artifacts (T1547). On Windows, check Run keys and scheduled tasks created by npm or VS Code plugin processes. For browser-based cryptocurrency wallet targeting (T1176), inventory installed browser extensions on developer machines and flag any not on an approved list. IOCs published in source reporting (Snyk blog, The Hacker News) should be ingested into SIEM and EDR platforms for correlation; note source quality score of 0.64, treat IOCs as medium-confidence until validated internally. Reference NIST AU-6 (Audit Record Review, Analysis, and Reporting) and CIS 8.2 (Collect Audit Logs) for log coverage requirements.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`graph.microsoft.com`	Legitimate Microsoft service abused as C2 channel via Graph API by UNK_DeadDrop and Contagious Interview; anomalous IDE or npm process connections to this domain are a detection indicator, not a block target	MEDIUM
URL	`https://snyk.io/blog/cursor-ide-malware-extension-compromise-in-usd500k-crypto-heist/`	Snyk analysis of Cursor IDE malicious extension incident; contains package-level IOCs — retrieve current IOC list directly from source	MEDIUM
URL	`https://thehackernews.com/2026/06/north-korean-hackers-are-turning.html`	Primary campaign reporting covering Contagious Interview and UNK_DeadDrop operations across VS Code, npm, and GitHub	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1041** — Exfiltration Over C2 Channel
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1608.001** — Upload Malware
- **T1102.002** — Bidirectional Communication
- **T1204.002** — Malicious File
- **T1071.001** — Web Protocols
- **T1555** — Credentials from Password Stores
- **T1059.005** — Visual Basic
- **T1059.007** — JavaScript
- **T1539** — Steal Web Session Cookie
- **T1176** — Software Extensions
- **T1102** — Web Service
- **T1566.002** — Spearphishing Link
- **T1547** — Boot or Logon Autostart Execution
- **T1036.005** — Match Legitimate Resource Name or Location
- **T1027** — Obfuscated Files or Information
- **T1059.004** — Unix Shell

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring

- **AT-2** — Literacy Training and Awareness
- **SI-3** — Malicious Code Protection
- **SI-8** — Spam Protection
- **CM-7** — Least Functionality
- **SI-7** — Software, Firmware, and Information Integrity
- **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
- **6.3** — Require MFA for Externally-Exposed Applications
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks
- **15.1** — Establish and Maintain an Inventory of Service Providers

HIPAA-SECURITY

- **164.312(d)** — Person or Entity Authentication
- **164.308(a)(5)(i)** — Security Awareness and Training

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.5.34** — Privacy and protection of personal information
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1041	Exfiltration Over C2 Channel	Exfiltration
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1608.001	Upload Malware	Resource-Development
T1102.002	Bidirectional Communication	Command-And-Control

Technique ID	Technique Name	Tactic
T1204.002	Malicious File	Execution
T1071.001	Web Protocols	Command-And-Control
T1555	Credentials from Password Stores	Credential-Access
T1059.005	Visual Basic	Execution
T1059.007	JavaScript	Execution
T1539	Steal Web Session Cookie	Credential-Access
T1176	Software Extensions	Persistence
T1102	Web Service	Command-And-Control
T1566.002	Spearphishing Link	Initial-Access
T1547	Boot or Logon Autostart Execution	Persistence
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion
T1027	Obfuscated Files or Information	Defense-Evasion
T1059.004	Unix Shell	Execution

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/06/north-korean-hackers-are-turning....	T3
Someone just lost $500000 for using cursor extensions. - Reddit	https://www.reddit.com/r/vscode/comments/1lxrxp9/someone_just_lost_...	T3
Cursor IDE Malware Extension Compromise in $500k Crypto Heist	https://snyk.io/blog/cursor-ide-malware-extension-compromise-in-usd...	T3
Malicious VS Code Extensions: Protect Developer Workstations	https://wizardcyber.com/malicious-vscode-extensions-threat/	T3
VS Code extension marketplace wars: Cursor users hit roadblocks	https://www.devclass.com/development/2025/04/08/vs-code-extension-m...	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 2026-06-16 07:11 UTC by TJS Security Command Center