INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-27 06:37 UTC

Glassworm Botnet Dismantled: How Developers Became the Supply Chain's Weakest Link

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0368
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	VSCode, Cursor, Positron, Windsurf, VSCodium extensions; OpenVSX marketplace; npm registry; PyPI; GitHub repositories; Node.js environments; Windows, macOS, Linux
Discovery Source	Rss:T1 Threatintel

Executive Summary

A Russia-linked threat group operating the Glassworm botnet spent over a year systematically compromising software developers by distributing malicious code extensions, poisoned open-source packages, and harvesting credentials from developer repositories. The campaign targeted the software supply chain directly, using developer workstations and CI/CD pipelines as entry points into downstream organizations. Organizations whose development teams use VSCode-compatible editors, npm, or PyPI packages face elevated risk of upstream code compromise, credential theft, and potential backdoors embedded in internally developed software.

Technical Analysis

Glassworm is a Russia-linked botnet dismantled May 26, 2026 via coordinated action by CrowdStrike Counter Adversary Operations, Google, and Shadowserver Foundation, with identified C2 infrastructure dismantled simultaneously. The campaign operated for more than a year across three primary attack vectors: (1) trojanized extensions distributed through OpenVSX and compatible marketplaces targeting VSCode, Cursor, Positron, Windsurf, and VSCodium; (2) poisoned packages seeded into npm and PyPI registries; (3) credential harvesting from GitHub repositories. No discrete CVE applies; the campaign exploited ecosystem trust and social engineering rather than a specific software vulnerability. Relevant CWEs: CWE-506 (Embedded Malicious Code), CWE-494 (Download of Code Without Integrity Check), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere). MITRE ATT&CK techniques include T1195.001 and T1195.002 (Supply Chain Compromise), T1176 (Browser/Editor Extensions), T1554 (Compromise Software Supply Chain), T1072 (Software Deployment Tools), T1555 and T1552 (Credential Access), T1568 and T1071.001 (C2 Communication), T1059.007 (JavaScript execution), and T1027 (Obfuscation). CVSS scoring does not apply to

campaign-level threat activity; qualitative rating (critical) is assigned editorially based on attack scope and supply chain impact. EPSS and CISA KEV data are not applicable.

Action Checklist

1. Step 1: Containment. Immediately audit all installed VSCode-compatible extensions across developer workstations for extensions sourced from OpenVSX or outside the official Microsoft Marketplace. Disable or quarantine unverified extensions pending review. Block developer systems from pulling new extensions or packages until inventory is complete. Enforce CIS 2.3 (Address Unauthorized Software) and CIS 1.2 (Address Unauthorized Assets).
2. Step 2: Detection. Query endpoint logs and EDR telemetry for outbound C2 beacon patterns, unexpected JavaScript process spawns (T1059.007), and connections to unknown external hosts from IDE processes (Code.exe, Cursor.exe, node.js). Audit npm and PyPI dependency trees in all active projects for packages with low download counts, recent publication dates, or mismatched checksums. Review GitHub access logs (per NIST AU-2, AU-6) for unauthorized commits, token usage, or repository cloning by unfamiliar identities. Check for system file modifications (per NIST SI-7) on developer hosts, including modified startup configurations.
3. Step 3: Eradication. Remove all flagged extensions, purge poisoned packages from project dependency files and lock files, and rotate all GitHub personal access tokens, SSH keys, and CI/CD secrets that were accessible from any potentially compromised developer workstation. Apply NIST SI-7 integrity verification controls to source code and build artifacts. Remove any unauthorized accounts or elevated permissions added via T1098 (Account Manipulation).
4. Step 4: Recovery. Re-build affected developer environments from clean images. Validate all code committed during the campaign window against known-good baselines before merging or deploying. Enable audit logging per NIST AU-3 and AU-12 across CI/CD systems. Implement multi-factor authentication on all GitHub, npm, and package registry accounts per CIS 6.3 and CIS 6.5. Verify no backdoored code was shipped to downstream products or customers.
5. Step 5: Post-Incident. Conduct a supply chain risk review against NIST SP 800-161r1 (Cyber Supply Chain Risk Management). Formalize an approved extension and package allowlist (NIST CM-7, Least Functionality; CIS 2.1, Software Inventory). Establish integrity verification for all third-party code ingestion (CWE-494 remediation) using checksum validation and signed package enforcement. Brief development leadership on software supply chain compromise patterns consistent with T1195.001 and T1195.002.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and external IR retainer immediately if forensic review confirms Glassworm-injected code was shipped in any customer-facing product release, any published open-source package, or any internal artifact consumed by downstream business units — each condition independently triggers potential breach notification obligations under GDPR, CCPA, or sector-specific regulations, and exceeds the response capability of a standard IR team given the Russia-linked attribution and active botnet infrastructure.

Recovery Notes	Treat the entire campaign window (minimum 12 months prior to detection) as compromised for all developer workstations that had any OpenVSX-sourced or unverified extension installed — do not selectively remediate only flagged systems, as Glassworm's lateral movement through CI/CD pipelines means a single infected workstation may have seeded multiple build environments. Monitor all newly rebuilt developer systems and CI/CD pipelines for 90 days post-recovery using Sysmon network connection logging on IDE and node processes, with daily review of GitHub audit logs for anomalous PAT usage or push events from unexpected IPs. Before resuming any package publishing activity to npm, PyPI, or other registries, perform a full integrity comparison of the to-be-published artifact against your source-controlled build pipeline output to confirm no Glassworm-backdoored code persists in the release candidate.
Forensic Artifacts	VSCode-compatible extension directories (~/.vscode/extensions/, %USERPROFILE%\.vscode\extensions\, and IDE-equivalent paths for Cursor, Windsurf, VSCodium, Positron) — each extension's package.json and bundled JavaScript files are the primary payload delivery mechanism and must be preserved before any quarantine action | Sysmon Event ID 3 (Network Connection) and Event ID 1 (Process Creation) logs filtered on Code.exe, Cursor.exe, node.exe, and windsurf.exe parent/child relationships — these establish Glassworm's T1059.007 JavaScript execution chain and C2 beacon timing patterns unique to IDE-hosted malicious extensions | npm package-lock.json and yarn.lock files, PyPI requirements.txt with hashes, and local package cache directories (~/.npm/, ~/.cache/pip/) — these record the exact poisoned package versions and checksums ingested during the campaign and are required to determine full dependency tree contamination scope | GitHub audit log exports covering the full campaign window including personal_access_token.create, git.push, deploy_key.create, and repository.fork events — Glassworm's credential harvesting from developer environments would surface as authentication events from IPs inconsistent with developer geolocation or as API calls at hours outside normal working patterns | CI/CD pipeline execution logs (GitHub Actions run history, .github/workflows/ YAML files, environment variable configurations) — Glassworm's use of developer workstations as entry points into downstream organizations would manifest as injected build steps, added outbound network calls in build logs, or modified workflow files that establish persistence in the build pipeline itself

Per-Action IR Details

Step 1: Containment — Immediately audit all installed VSCode-compatible extensions across developer workstations for extensions sourced from OpenVSX or outside the official Microsoft Marketplace. Disable or quarantine unverified extensions pending review. Block developer systems from pulling new extensions or packages until inventory is complete. Enforce CIS 2.3 (Address Unauthorized Software) and CIS 1.2 (Address Unauthorized Assets).

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), NIST CM-8 (System Component Inventory), CIS 2.3 (Address Unauthorized Software), CIS 1.2 (Address Unauthorized Assets), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: On Windows, enumerate all installed VSCode-compatible extensions by running: `Get-ChildItem -Path "$env:USERPROFILE\.vscode\extensions" -Directory | Select-Object Name` and repeat for Cursor (`%APPDATA%\Cursor\User\extensions`), Windsurf, and VSCodium equivalents. On macOS/Linux: `ls ~/.vscode/extensions/ ~/.cursor/extensions/` for each IDE. Cross-reference output against a manually curated allowlist of known-safe publisher IDs. Use `jq` to parse each extension's `package.json` for `publisher`, `repository`, and any `scripts` fields that invoke network calls. Block outbound TCP 443 at the host firewall using `ufw deny out 443` or

Windows Defender Firewall rules scoped to Code.exe, Cursor.exe, and node.exe until inventory is complete.

Evidence: Before quarantining any extension, preserve the full extension directory (e.g., `~/.vscode/extensions/.-/`) including `package.json`, `extension.js`, and any bundled `.vsix` files — these contain the malicious payload delivered by Glassworm. Capture a filesystem snapshot or use `tar czf extensions_backup.tar.gz ~/.vscode/extensions/` before removal. On Windows, capture `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` and `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run` for any extension-injected persistence. Collect process tree snapshots showing active node.exe or Code.exe child processes via `Get-CimInstance Win32_Process | Where-Object {$_.Name -match 'node|code|cursor'} | Select-Object ProcessId,ParentProcessId,CommandLine`.

Step 2: Detection — Query endpoint logs and EDR telemetry for outbound C2 beacon patterns, unexpected JavaScript process spawns (T1059.007), and connections to unknown external hosts from IDE processes (Code.exe, Cursor.exe, node.js). Audit npm and PyPI dependency trees in all active projects for packages with low download counts, recent publication dates, or mismatched checksums. Review GitHub access logs (AU-2, AU-6) for unauthorized commits, token usage, or repository cloning by unfamiliar identities. Check for D3-SFA indicators: modified system files or startup configs (D3-SICA) on developer hosts.

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), NIST SI-3 (Malicious Code Protection), NIST RA-5 (Vulnerability Monitoring and Scanning), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Deploy Sysmon with a configuration that logs Event ID 3 (Network Connection) filtered on Image paths matching `*\Code.exe`, `*\Cursor.exe`, `*\node.exe`, and `*\windsurf.exe` — any outbound connection from these processes to non-CDN, non-Microsoft IPs is a Glassworm C2 indicator. Use Sigma rule 'proc_creation_win_node_js_exec_from_vscode' as a template for detecting T1059.007. For npm: run `npm audit --json` and `npx lockfile-lint` in each project root; flag packages published within 90 days of the campaign window (2023–2024) with fewer than 500 weekly downloads. For PyPI: use `pip-audit` (`pip install pip-audit && pip-audit`) and cross-reference `requirements.txt` hashes against PyPI's JSON API (`https://pypi.org/pypi///json`). Query GitHub audit logs via API: `gh api /orgs//audit-log --paginate | jq '.[] | select(.action=="git.push" or .action=="personal_access_token.create")'` filtering for actor usernames and IPs not matching known developer baselines.

Evidence: Capture Sysmon Event ID 3 logs showing outbound connections from IDE processes before blocking egress — these establish C2 infrastructure attribution. Preserve npm lock files (`package-lock.json`, `yarn.lock`) and PyPI `requirements.txt` with intact checksums before any remediation; these are the forensic record of what poisoned packages were ingested. Export GitHub audit log for the full campaign window (minimum 12 months given Glassworm's operational timeline) in JSON format, preserving PAT creation events, push events from unfamiliar IPs, and any `deploy_key.create` actions. On Linux/macOS, collect `~/.bash_history`, `~/.zsh_history`, and `~/.npm/_logs/` for evidence of developer-executed install commands that pulled Glassworm-affiliated packages.

Step 3: Eradication — Remove all flagged extensions, purge poisoned packages from project dependency files and lock files, and rotate all GitHub personal access tokens, SSH keys, and CI/CD secrets that were accessible from any potentially compromised developer workstation (D3-CRO — Credential Rotation). Apply NIST SI-7 integrity verification controls to source code and build artifacts. Remove any unauthorized accounts or elevated permissions added via T1098 (Account Manipulation).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-7 (Software, Firmware, and Information Integrity), NIST IA-5 (Authenticator Management), NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), NIST CM-3 (Configuration Change Control), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 5.3 (Disable Dormant Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: For credential rotation without a secrets manager: enumerate all GitHub PATs via `gh api /user/tokens` and revoke all tokens with scopes beyond `read:user` created before the confirmed clean date, then

reissue with minimum necessary scope and expiration. Rotate SSH keys by removing all entries from `~/.ssh/authorized_keys` and `GitHub → Settings → SSH Keys` and reissuing Ed25519 keys (`ssh-keygen -t ed25519`). For CI/CD secrets in GitHub Actions: use `gh secret list` to inventory all repository and org-level secrets, then rotate each one in the upstream provider before updating in GitHub. To verify integrity of source code per SI-7: run `git log --all --oneline --format='%H %ae %ai %s'` for the campaign window and manually review commits from any email address or timestamp pattern inconsistent with your team's known working hours and identities. Use `git verify-commit ` to check GPG signature presence on commits during the suspect window.

Evidence: Before revoking PATs and SSH keys, export the full list of active tokens and their last-used timestamps from GitHub (`gh api /user/tokens --jq '.[] | {id,description,last_used_at,scopes}'`) — this establishes whether Glassworm actors actively used harvested credentials. Preserve CI/CD pipeline execution logs (GitHub Actions logs, `.github/workflows/` run history) for the campaign window showing any unexpected workflow triggers, added steps, or exfiltration-suggestive outbound calls in build logs. Capture the git reflog (`git reflog --all`) before any branch cleanup — Glassworm-inserted commits may have been force-pushed over or hidden in detached HEAD states. On developer hosts, collect `~/.ssh/known_hosts` and `~/.gitconfig` for any attacker-modified remote URLs redirecting pushes to adversary-controlled repositories.

Step 4: Recovery — Re-build affected developer environments from clean images. Validate all code committed during the campaign window against known-good baselines before merging or deploying. Enable audit logging per AU-3 and AU-12 across CI/CD systems. Implement D3-MFA (Multi-factor Authentication) on all GitHub, npm, and package registry accounts per CIS 6.3 and CIS 6.5. Verify no backdoored code was shipped to downstream products or customers.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-3 (Content of Audit Records), NIST AU-12 (Audit Record Generation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CP-10 (System Recovery and Reconstitution), NIST IA-2 (Identification and Authentication — Organizational Users), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Rebuild developer workstations from vendor-supplied OS images or a documented golden image, then reinstall only allowlisted IDE extensions by maintaining a controlled `.vsix` file repository (downloaded from the official Microsoft Marketplace and hash-verified using `sha256sum`) — do not permit reinstallation from OpenVSX or arbitrary sources. For code validation without a commercial SAST tool: run `git diff ..HEAD` across all active branches covering the campaign window, then manually review any changes to `package.json`, `setup.py`, build scripts, workflow YAML files, and any files that establish network connections or handle credentials. Enable GitHub audit log streaming to a persistent destination (GitHub supports streaming to S3 or Splunk — for budget-constrained teams, use `gh api /orgs//audit-log --paginate > audit_log_$(date +%F).json` as a daily cron job). Enforce MFA via GitHub org policy: `Settings → Authentication security → Require two-factor authentication`.

Evidence: Before rebuilding workstations, acquire a full forensic image of each affected system using `dc3dd` or `dd` — do not rely solely on logical copies, as Glassworm's implants may have modified startup scripts (`~/.bashrc`, `~/.zshrc`, `/etc/profile.d/`, Windows `Shell:Startup`) that would be missed by file-level collection. Capture all npm and PyPI cache directories (`~/.npm/`, `~/.cache/pip/`) which may contain downloaded-but-not-installed poisoned package tarballs that establish the full scope of attempted supply chain injection. For downstream impact verification: extract the published artifact manifests (npm `package.json` dist-tags, PyPI release files) for any packages your organization publishes and diff them against your source-controlled build outputs to detect Glassworm-injected code that may have been published to the registry during the compromise window.

Step 5: Post-Incident — Conduct a supply chain risk review against NIST SP 800-161r1 (Cyber Supply Chain Risk Management). Formalize an approved extension and package allowlist (NIST CM-7, Least Functionality; CIS 2.1, Software Inventory). Establish integrity verification for all third-party code ingestion (CWE-494 remediation) using checksum validation and signed package enforcement. Brief development leadership on software supply chain compromise patterns consistent with T1195.001 and T1195.002.

NIST Phase: Post Incident

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

Controls: NIST CM-7 (Least Functionality), NIST CM-8 (System Component Inventory), NIST SA-12 (Supply Chain Protection), NIST RA-3 (Risk Assessment), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Formalize the extension allowlist as a JSON file committed to a protected branch in your internal repository, containing approved publisher IDs and extension versions with their SHA-256 hashes sourced from the official marketplace — enforce this via a pre-commit hook or CI check that runs `code --list-extensions` and diffs against the allowlist on each developer machine check-in. For package integrity enforcement without a commercial SCA tool: configure npm's built-in `npm config set ignore-scripts true` org-wide to prevent malicious `postinstall` hooks (a documented Glassworm delivery mechanism), enable `npm config set audit true`, and adopt `pip install --require-hashes -r requirements.txt` for PyPI to mandate hash pinning. Document T1195.001 (Compromise Software Dependencies and Development Tools) and T1195.002 (Compromise Software Supply Chain) as named threat scenarios in your threat model, with specific detection hypotheses tied to IDE process network behavior and package registry anomalies surfaced during this incident.

Evidence: Compile the full lessons-learned artifact package for the supply chain risk review: the extension inventory snapshots from Step 1, the poisoned package list from Step 2, the git audit from Step 3, and the downstream artifact diff from Step 4 — together these establish the complete Glassworm kill chain traversal through your environment. Preserve all Glassworm-affiliated extension `.vsix` files and npm/PyPI package tarballs in an isolated evidence archive (write-once storage or a hashed ZIP with chain-of-custody documentation) for potential law enforcement referral given the Russia-linked attribution and potential downstream victim notification obligations. Document any customer-facing products or open-source packages published during the campaign window that may have contained Glassworm-injected code, as these establish the scope of required downstream breach notification.

Detection Guidance

Primary detection focus: suspicious IDE and Node.js process behavior, outbound C2 traffic from developer hosts, and dependency integrity anomalies. Key indicators and queries: (1) EDR/endpoint: alert on child processes spawned by Code.exe, Cursor.exe, or node.exe making outbound network connections to non-CDN, non-registry IPs, especially on non-standard ports. (2) Network logs: look for periodic low-volume beaconing (consistent interval, small payload) from developer subnets consistent with T1568 dynamic resolution and T1071.001 HTTP-based C2. (3) Package integrity: diff current npm package-lock.json and PyPI requirements files against last-known-good versions; flag packages added or modified during the campaign window (pre-May 26, 2026). Use 'npm audit' and verify package checksums against registry hashes. (4) GitHub audit logs: review for OAuth token grants, unexpected repository forks, unusual commit authors, or API access from unrecognized IPs during the campaign period. (5) Extension inventory: enumerate all installed extensions via 'code --list-extensions' across all developer hosts; cross-reference against a verified allowlist. Flag any extension sourced from OpenVSX that is not explicitly approved. (6) System file analysis: monitor for modified shell profiles (.bashrc, .zshrc), SSH authorized_keys, or npm/pip configuration files that could establish persistence. (7) Local account monitoring: check for new local accounts or privilege escalations on developer workstations. CWE-506 artifacts: scan build outputs and committed code for obfuscated blobs or encoded strings consistent with embedded malicious code (T1027).

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	[not publicly released at time of analysis]	C2 infrastructure domains used by Glassworm botnet — not disclosed in available source data at time of writing; monitor CrowdStrike threat intelligence feeds for IOC release	LOW
HASH	[not publicly released at time of analysis]	Malicious extension and package file hashes — not disclosed in available source data; expected in follow-on CrowdStrike and Shadowserver disclosures	LOW

Framework Mappings

MITRE-ATTACK

- **T1568.003** — DNS Calculation
- **T1071.001** — Web Protocols
- **T1195.002** — Compromise Software Supply Chain
- **T1102.003** — One-Way Communication
- **T1555** — Credentials from Password Stores
- **T1554** — Compromise Host Software Binary
- **T1098** — Account Manipulation
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1568** — Dynamic Resolution
- **T1566** — Phishing
- **T1552** — Unsecured Credentials
- **T1072** — Software Deployment Tools
- **T1588.001** — Malware
- **T1059.007** — JavaScript
- **T1041** — Exfiltration Over C2 Channel
- **T1027** — Obfuscated Files or Information
- **T1176** — Software Extensions
- **T1102** — Web Service
- **T1078** — Valid Accounts

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
- **AT-2** — Literacy Training and Awareness

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-8** — Spam Protection
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **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

SOC2-TSC

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

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

NIST-CSF-2

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1568.003	DNS Calculation	Command-And-Control
T1071.001	Web Protocols	Command-And-Control

Technique ID	Technique Name	Tactic
T1195.002	Compromise Software Supply Chain	Initial-Access
T1102.003	One-Way Communication	Command-And-Control
T1555	Credentials from Password Stores	Credential-Access
T1554	Compromise Host Software Binary	Persistence
T1098	Account Manipulation	Persistence
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1568	Dynamic Resolution	Command-And-Control
T1566	Phishing	Initial-Access
T1552	Unsecured Credentials	Credential-Access
T1072	Software Deployment Tools	Execution
T1588.001	Malware	Resource-Development
T1059.007	JavaScript	Execution
T1041	Exfiltration Over C2 Channel	Exfiltration
T1027	Obfuscated Files or Information	Defense-Evasion
T1176	Software Extensions	Persistence
T1102	Web Service	Command-And-Control
T1078	Valid Accounts	Defense-Evasion

Sources

Source	URL	Tier
Blog	https://www.crowdstrike.com/en-us/blog/inside-crowdstrike-takedown-...	T3
Critical Open VSX Registry Flaw Exposes Millions of ...	https://thehackernews.com/2025/06/critical-open-vsx-registry-flaw-e...	T3
Hunting Malicious VS Code Extensions (GlassWorm ...	https://wizardcyber.com/malicious-vscode-extensions-threat/	T3
VSCodium	https://github.com/VSCodium/vscodium	T3

Source	URL	Tier
Inside CrowdStrike's Takedown of a Developer-Targeting ...	https://www.crowdstrike.com/content/crowdstrike-www/locale-sites/us...	T3

Generated 2026-05-27 06:37 UTC by TJS Security Command Center