INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-30 19:07 UTC

CrowdStrike, Google, and Shadowserver Dismantle Glassworm Developer-Targeting Botnet

THREAT CAMPAIGN | **HIGH** | CVSS 8.1

SCC Item ID	SCC-CAM-2026-0386
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	8.1
Affected Products	Developers on Windows, macOS, and Linux; GitHub repositories; CI/CD pipelines; IDEs including Cursor, Positron, Windsurf, and VSCodium; npm and Python package ecosystems
Published	2026-05-30
Discovery Source	Gemini

Executive Summary

CrowdStrike, Google, and the Shadowserver Foundation dismantled Glassworm, a botnet that targeted software developers by distributing malware through compromised GitHub repositories, malicious IDE extensions for Cursor, Positron, Windsurf, and VSCodium, and trojanized npm and Python packages. The campaign ran across Windows, macOS, and Linux, using four separate command-and-control channels to maintain resilience. The strategic objective was downstream access, compromising developer infrastructure to reach the organizations and end users those developers build for, making every application in an affected pipeline a potential victim.

Technical Analysis

Glassworm is a developer-targeting botnet takedown attributed to a joint operation by CrowdStrike, Google, and Shadowserver Foundation. Attack vectors included compromised GitHub repositories (T1195.001, Supply Chain Compromise: Compromise Software Dependencies and Development Tools), malicious VSCode-compatible IDE extensions targeting Cursor, Positron, Windsurf, and VSCodium (T1176, Browser/Extension Compromise used in IDE context), and trojanized npm and Python packages (T1195.002, Supply Chain Compromise: Compromise Software Supply Chain). Payload delivery leveraged T1059 (Command and Scripting Interpreter) and T1072 (Software Deployment Tools). The botnet used dynamic resolution and four distinct C2 channels (T1071, Application Layer Protocol; T1568, Dynamic Resolution) to complicate takedown and maintain persistence. Relevant CWEs: CWE-912 (Hidden Functionality), CWE-494 (Download of Code Without Integrity

Check), CWE-829 (Inclusion of Functionality from Untrusted Control Sphere), CWE-506 (Embedded Malicious Code). No CVE identifiers are associated with this campaign. Technical specifics carry medium confidence pending direct validation against CrowdStrike's primary technical reporting. The takedown has been executed; however, residual infections and copycat packages may persist.

Action Checklist

1. Step 1: Containment. Audit all IDE extensions installed in developer environments, specifically for Cursor, Positron, Windsurf, and VSCodium. Disable or quarantine any extension not sourced from a verified, internally approved registry. Enforce CIS 2.3 (Address Unauthorized Software) by blocking unapproved extensions at the endpoint level. Validate GitHub repository integrity for any repos cloned or forked during the suspected campaign window.
2. Step 2: Detection. Review endpoint logs for anomalous outbound connections from IDE processes, npm install operations, or Python pip invocations. Look for script interpreter spawning (cmd.exe, powershell.exe, bash, python) as child processes of IDE executables, consistent with T1059 execution chains. Audit npm and pip install history for packages not present in your internal approved inventory (NIST AU-2, CIS 8.2). Query DNS logs for domains resolved by developer workstations that do not match known CDN or package registry infrastructure. Apply File Integrity Monitoring (FIM) to monitor IDE extension directories and package cache locations for unauthorized modifications.
3. Step 3: Eradication. Remove any flagged IDE extensions and re-install only from verified sources with hash verification enabled (NIST CM controls). Purge compromised npm and Python packages from all developer environments and CI/CD pipeline caches. Rotate all credentials, tokens, and API keys accessible from affected developer workstations; treat all secrets on compromised machines as exposed. Re-clone GitHub repositories from known-good commits with verified commit signatures where possible.
4. Step 4: Recovery. Validate CI/CD pipeline integrity end-to-end before resuming production builds. Confirm all build artifacts produced during the campaign window are re-built from clean sources. Monitor developer endpoints and pipeline systems for 30 days post-remediation for re-infection indicators using behavioral detection and service account monitoring (NIST AU-2, AU-6). Verify outbound C2 channel indicators are blocked at the network perimeter (NIST SC-7, Boundary Protection).
5. Step 5: Post-Incident. This campaign exposes gaps in software supply chain integrity verification. Implement NIST SI-7 (Software, Firmware, and Information Integrity) controls, specifically integrity verification for packages, extensions, and third-party dependencies. Establish an approved software inventory (CIS 2.1) that includes IDE extensions and package dependencies. Require cryptographic signing and hash verification for all packages ingested into CI/CD pipelines (CWE-494 mitigation). Conduct a threat hunt for related indicators across developer workstations and build servers using the MITRE ATT&CK techniques mapped to this campaign.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO and legal counsel immediately if forensic analysis confirms any CI/CD build artifacts produced during the Glassworm campaign window were distributed to external customers or downstream open-source consumers, triggering potential software supply chain breach notification obligations; also escalate if GitHub organization audit logs reveal Glassworm actors added unauthorized collaborators or exfiltrated repository secrets.
Recovery Notes	Before resuming any production builds, re-hash and re-validate all build artifacts produced during the campaign window against pre-campaign baseline hashes — any mismatch must trigger downstream customer notification assessment given Glassworm's explicit supply chain contamination objective. Monitor all developer endpoints and CI/CD pipeline service accounts for 30 days post-remediation using osquery process tree queries and DNS blocklist hit logs, specifically watching for re-emergence of IDE child process anomalies or outbound connections to newly registered domains not matching known package registry infrastructure. Given Glassworm's four-channel C2 resilience design, confirm all four channel types (not just the primary IOC) are blocked at DNS, firewall, and proxy layers before declaring containment complete.
Forensic Artifacts	IDE extension directories for all four targeted IDEs (Cursor: %USERPROFILE%\.cursor\extensions on Windows / ~/.cursor/extensions on Linux; Positron: ~/.positron/extensions; Windsurf: ~/.windsurf/extensions; VSCodium: ~/.vscode-oss/extensions) — Glassworm's primary delivery mechanism; trojanized extension package.json and main .js entry-point files will show hash mismatches against vendor-published versions | npm package cache (~/.npm/_logs/ for install logs; node_modules//package.json for installed package metadata including _integrity field) — records the exact Glassworm-affiliated malicious package names, versions, and install timestamps; _integrity SHA-512 mismatches confirm tampered packages | Sysmon Event ID 1 (Process Creation) and Event ID 3 (Network Connection) logs filtered on IDE parent process names (cursor.exe, codium.exe, windsurf.exe) — captures the specific T1059 execution chains Glassworm extensions triggered and the four distinct C2 outbound connection patterns unique to this botnet's multi-channel architecture | GitHub organization audit log (JSON export from Settings → Audit Log) covering the campaign window — records repository forks, collaborator additions, Actions workflow modifications, and Secrets access events that indicate Glassworm's downstream supply chain compromise attempts beyond the initial developer workstation infection | Shell history files (~/.bash_history, ~/.zsh_history) and Python pip install logs (~/.local/lib/python*/site-packages/ directory with inode timestamps) on developer workstations — captures the exact Glassworm-affiliated package names ingested via pip during the campaign window, including packages that may have been subsequently removed to evade detection

Per-Action IR Details

Step 1: Containment — Audit all IDE extensions installed in developer environments, specifically for Cursor, Positron, Windsurf, and VSCodium. Disable or quarantine any extension not sourced from a verified, internally approved registry. Enforce CIS 2.3 (Address Unauthorized Software) by blocking unapproved extensions at the endpoint level. Validate GitHub repository integrity for any repos cloned or forked during the campaign window.

NIST Phase: Containment

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

Controls: NIST CM-7 (Least Functionality) — restrict IDE extension installation to approved sources only, NIST CM-8 (System Component Inventory) — enumerate all installed extensions across Cursor, Positron, Windsurf, and VSCodium on developer endpoints, NIST SI-3 (Malicious Code Protection) — quarantine flagged extensions before removal, CIS 2.3 (Address Unauthorized Software) — remove or exception-document any extension absent from

internal approved inventory, CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory) — confirm all developer workstations are in scope for the audit

Compensating: Run the following PowerShell one-liner on Windows developer endpoints to enumerate all VSCodium/Cursor/Windsurf extensions: `Get-ChildItem -Path "$env:USERPROFILE\.vscode-oss\extensions","$env:USERPROFILE\.cursor\extensions","$env:USERPROFILE\.windsurf\extensions" -Directory | Select-Object Name,LastWriteTime | Export-Csv extensions_audit.csv`. On macOS/Linux, run: `find ~/.vscode-oss/extensions ~/.cursor/extensions ~/.windsurf/extensions -maxdepth 1 -type d 2>/dev/null`. Diff the output against your approved extension list; flag any extension installed or modified during the Glassworm campaign window. Use `git log --oneline --since=` on all locally cloned repos and cross-reference commit hashes against upstream GitHub to detect tampered commits injected via Glassworm-compromised repo access.

Evidence: BEFORE disabling any extensions, capture: (1) Full directory listing with timestamps from all four IDE extension directories (~/.cursor/extensions, ~/.positron/extensions, ~/.windsurf/extensions, ~/.vscode-oss/extensions on Linux/macOS; %USERPROFILE%\.cursor\extensions on Windows). (2) SHA-256 hashes of all extension package.json and main entry-point .js files — Glassworm trojanized extensions will show hash mismatches against known-good versions in the VSCodium/Cursor marketplace. (3) Git reflog and `git log --all --pretty=format:"%H %ae %s"` from any repos cloned during the campaign window to identify unauthorized commits. (4) Network connection state at time of discovery: `netstat -anb` (Windows) or `ss -tulnp` (Linux) filtered for IDE process PIDs to capture live C2 connections before containment severs them.

Step 2: Detection — Review endpoint logs for anomalous outbound connections from IDE processes, npm install operations, or Python pip invocations. Look for script interpreter spawning (cmd.exe, powershell.exe, bash, python) as child processes of IDE executables — consistent with T1059 execution chains. Audit npm and pip install history for packages not present in your internal approved inventory (NIST AU-2, CIS 8.2). Query DNS logs for domains resolved by developer workstations that do not match known CDN or package registry infrastructure. Apply D3-SFA (System File Analysis) to monitor IDE extension directories and package cache locations for unauthorized modifications.

NIST Phase: Detection Analysis

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

Controls: NIST AU-2 (Event Logging) — ensure IDE process execution, child process creation, and network connections are captured in endpoint logs, NIST AU-6 (Audit Record Review, Analysis, and Reporting) — review logs for Glassworm-specific execution chains: IDE parent → script interpreter child, NIST SI-4 (System Monitoring) — monitor developer endpoints for anomalous outbound connections from Cursor, Positron, Windsurf, and VSCodium processes, CIS 8.2 (Collect Audit Logs) — validate that npm install and pip invocation events are captured on developer workstations, MITRE ATT&CK T1059 (Command and Scripting Interpreter) — detect script interpreter spawning from IDE extension execution, MITRE ATT&CK T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools) — scope detection to package manager invocations, MITRE ATT&CK T1071 (Application Layer Protocol) — identify Glassworm's four C2 channels communicating over standard application protocols to evade detection

Compensating: Deploy Sysmon with a config including ProcessCreate (Event ID 1) rules targeting parent process image paths matching cursor.exe, positron.exe, windsurf.exe, or codium.exe with child processes of cmd.exe, powershell.exe, bash, python, or node.exe. Use this Sysmon filter: `cursor;positron;windsurf;codium`. Query collected Sysmon logs with: `Get-WinEvent -LogName "Microsoft-Windows-Sysmon/Operational" | Where-Object {$_.Id -eq 1} | Where-Object {$_.Message -match "cursor|windsurf|positron|codium"} | Select-Object TimeCreated,Message`. On Linux, use auditd rules: `auditctl -a always,exit -F arch=b64 -S execve -F ppid=$(pgrep -d, -x cursor) -k glassworm_exec`. For DNS analysis without SIEM, run `python3 -m http.server` or parse /var/log/named or Windows DNS debug logs filtering for non-registry domains: legitimate npm uses registry.npmjs.org and pypi.org — flag everything else resolved by IDE or package manager processes.

Evidence: BEFORE concluding detection scope: (1) Windows Security Event Log Event ID 4688 (Process Creation) filtered on creator process name matching IDE executables — captures script interpreter spawning chains specific to Glassworm extension payload execution. (2) Sysmon Event ID 3 (Network Connection) filtered on IDE process PIDs — Glassworm's four-channel C2 architecture means developer workstations will show outbound connections to multiple distinct C2 endpoints, not just one. (3) npm install log at ~/.npm/_logs/ (Linux/macOS) or

%APPDATA%\npm-cache_logs\ (Windows) — records exact package names, versions, and install timestamps for packages pulled during the campaign window. (4) pip install history via `pip list --format=json` cross-referenced against ~/.local/lib/python*/site-packages/ directory timestamps. (5) DNS query logs (Windows: Event ID 3008 in Microsoft-Windows-DNS-Client/Operational) filtered by process to isolate domains resolved exclusively by IDE or package manager processes — Glassworm C2 domains will not match npmjs.org, pypi.org, github.com, or known CDN ranges.

Step 3: Eradication — Remove any flagged IDE extensions and re-install only from verified sources with hash verification enabled (NIST CM controls; D3-FMBV — File Magic Byte Verification). Purge compromised npm and Python packages from all developer environments and CI/CD pipeline caches. Rotate all credentials, tokens, and API keys accessible from affected developer workstations — treat all secrets on compromised machines as exposed (D3-CRO — Credential Rotation). Re-clone GitHub repositories from known-good commits with verified commit signatures where possible.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST CM-3 (Configuration Change Control) — document and track all extension removals and re-installations against approved baseline, NIST CM-7 (Least Functionality) — re-establish approved extension baseline, removing all Glassworm-identified or unverifiable extensions, NIST IA-5 (Authenticator Management) — rotate all credentials and tokens accessible from workstations confirmed or suspected to be under Glassworm C2 control, NIST SI-2 (Flaw Remediation) — apply vendor-verified clean extension packages with hash validation before re-deployment, CIS 5.1 (Establish and Maintain an Inventory of Accounts) — enumerate all service accounts, CI/CD tokens, and API keys accessible from affected developer environments for targeted rotation, MITRE ATT&CK T1078 (Valid Accounts) — assume all tokens stored in IDE config files, .env files, shell history, and CI/CD environment variables on compromised hosts are stolen

Compensating: For hash verification without enterprise tooling: after downloading a clean extension VSIX from the vendor-verified source, run `certutil -hashfile SHA256` (Windows) or `sha256sum ` (Linux/macOS) and compare against the vendor-published hash. Install manually with `codium --install-extension `. For npm package verification: `npm install @ --dry-run` then verify with `npm audit` and cross-check package integrity using `cat node_modules//package.json | python3 -m json.tool | grep _integrity`. For credential rotation scope discovery on Linux: `grep -rE "(API_KEY|TOKEN|SECRET|PASSWORD|GITHUB_TOKEN)" ~/.bashrc ~/.zshrc ~/.env ~/.gitconfig ~/.*rc 2>/dev/null` and `env | grep -iE "key|token|secret"` to enumerate what Glassworm could have exfiltrated from the shell environment.

Evidence: BEFORE eradication removes artifacts: (1) Disk image or forensic copy of the IDE extension directory from each affected workstation — Glassworm's trojanized extension files are the primary malware artifact and must be preserved for signature extraction and YARA rule development. (2) Memory dump (using WinPmem on Windows or LiME on Linux) from any workstation with a confirmed live Glassworm C2 connection — in-memory artifacts may reveal decrypted C2 payload or stolen credential buffers not yet written to disk. (3) Full export of npm global cache (`npm cache ls --json`) and pip cache (`pip cache list`) before purging — preserves the exact malicious package versions for IOC sharing with CrowdStrike, Google, and Shadowserver as part of the coordinated dismantlement. (4) Git log with signatures from all repos cloned from Glassworm-compromised GitHub accounts: `git log --show-signature --pretty=format:"%H %G? %GS %ae %s"` — unsigned or invalidly-signed commits during the campaign window indicate Glassworm-injected code.

Step 4: Recovery — Validate CI/CD pipeline integrity end-to-end before resuming production builds. Confirm all build artifacts produced during the campaign window are re-built from clean sources. Monitor developer endpoints and pipeline systems for 30 days post-remediation for re-infection indicators using behavioral detection (D3-LAM — Local Account Monitoring for tokens and service accounts used in pipelines). Verify outbound C2 channel indicators are blocked at the network perimeter (NIST SC-7 — Boundary Protection).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SC-7 (Boundary Protection) — block confirmed Glassworm C2 infrastructure at perimeter firewall and DNS resolvers; given four-channel C2 architecture, blocking must cover all four identified channels, NIST SI-7 (Software, Firmware, and Information Integrity) — verify integrity of all build artifacts produced during the campaign window before promoting to production or distributing downstream, NIST CP-10 (System Recovery and Reconstitution) — recover CI/CD pipeline configuration from last known-good state prior to campaign window, NIST AU-12 (Audit Record Generation) — confirm logging is re-enabled and collecting on all recovered developer endpoints and CI/CD systems before declaring recovery complete, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — scan all recovered systems with updated signatures including Glassworm IOCs before returning to production, MITRE ATT&CK T1195.002 (Supply Chain Compromise: Compromise Software Supply Chain) — re-validate that no Glassworm-injected code reached downstream consumers via CI/CD artifacts built during the campaign window

Compensating: For CI/CD integrity validation without enterprise tooling: generate SHA-256 hashes of all pipeline configuration files (Jenkinsfile, .github/workflows/*.yml, .gitlab-ci.yml, Dockerfile) using `find . -name "*.yml" -o -name "Jenkinsfile" -o -name "Dockerfile" | xargs sha256sum > pipeline_hashes_$(date +%F).txt` and diff against hashes from your last pre-campaign Git commit. For 30-day re-infection monitoring without EDR, deploy osquery with a pack querying: `SELECT pid, name, parent, cmdline FROM processes WHERE parent IN (SELECT pid FROM processes WHERE name IN ('cursor','codium','windsurf','positron'));` on a 5-minute interval and pipe to a central log file. Block Glassworm C2 domains at the DNS level using a local Unbound or Pi-hole instance loaded with published IOC blocklists from CrowdStrike and Shadowserver's campaign disclosures.

Evidence: BEFORE resuming production builds: (1) Diff of all CI/CD pipeline YAML and configuration files between last known-good commit hash (pre-campaign) and current state — Glassworm targeting CI/CD pipelines means pipeline configuration modification is a high-probability artifact. (2) Build artifact hash comparison: re-hash all binaries, containers, and packages produced during the campaign window and compare against the hashes committed in your artifact registry or release notes — mismatches indicate Glassworm-injected build tampering. (3) Service account and CI/CD token usage logs from GitHub Actions audit log (available under Organization → Settings → Audit log) filtered for the campaign window — unusual workflow triggers, repository access from unexpected IP ranges, or secret access events indicate Glassworm lateral movement through pipeline credentials. (4) Network perimeter firewall logs confirming outbound block events to all four Glassworm C2 channel addresses — absence of post-remediation outbound attempts is a prerequisite for recovery sign-off.

Step 5: Post-Incident — This campaign exposes gaps in software supply chain integrity verification. Implement NIST SI-7 (Software, Firmware, and Information Integrity) controls — specifically integrity verification for packages, extensions, and third-party dependencies. Establish an approved software inventory (CIS 2.1) that includes IDE extensions and package dependencies. Require cryptographic signing and hash verification for all packages ingested into CI/CD pipelines (CWE-494 mitigation). Conduct a threat hunt for related indicators across developer workstations and build servers using the MITRE ATT&CK techniques mapped to this campaign.

NIST Phase: Post Incident

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

Controls: NIST SI-7 (Software, Firmware, and Information Integrity) — mandate hash and signature verification for all IDE extensions, npm packages, and Python dependencies ingested into developer environments and CI/CD pipelines, NIST CA-7 (Continuous Monitoring) — establish ongoing monitoring of IDE extension directories and package manager caches for unauthorized changes, informed by Glassworm TTPs, NIST RA-3 (Risk Assessment) — update organizational risk register to reflect developer toolchain and software supply chain as an elevated attack surface following Glassworm, NIST IR-4 (Incident Handling) — update IR playbook to include software supply chain compromise scenarios, specifically IDE extension and package manager compromise vectors, CIS 2.1 (Establish and Maintain a Software Inventory) — extend software inventory scope to explicitly include IDE extensions for Cursor, Positron, Windsurf, and VSCodium as managed software assets, CIS 7.1 (Establish and Maintain a Vulnerability Management Process) — incorporate package ecosystem monitoring (npm advisories, PyPI malware reports) into the vulnerability management process, MITRE ATT&CK T1195.001 (Supply Chain Compromise: Compromise Software Dependencies and Development Tools) — build detection rules in Sysmon and auditd tuned to the specific IDE process trees and package manager invocation patterns observed in this campaign

Compensating: For the threat hunt without a SIEM: use YARA rules targeting Glassworm extension signatures (publish rule targeting trojanized package.json patterns and obfuscated JS payloads in extension directories) and run with `yara -r glassworm.yar ~/.cursor/extensions ~/.windsurf/extensions ~/.vscode-oss/extensions`. For ongoing supply chain integrity: implement `npm ci` (clean install from lockfile) instead of `npm install` in all CI/CD pipelines — this enforces exact dependency resolution and detects lockfile tampering. Add `pip-audit` (free, from PyPA) to CI/CD pre-build steps: `pip-audit -r requirements.txt` scans for known-malicious packages. For cryptographic signing enforcement: configure git with `git config --global commit.gpgsign true` and add a GitHub Actions workflow step that rejects unsigned commits with `git log --show-signature HEAD | grep -q "Good signature" || exit 1`. Document all findings in a lessons-learned report and share relevant IOCs with Shadowserver and CrowdStrike via their established disclosure channels from the Glassworm dismantlement operation.

Evidence: For lessons-learned and ongoing hunt: (1) Consolidated timeline of all Glassworm-attributed events across the organization, built from DNS query logs, process creation logs, and extension modification timestamps — forms the basis for detection rule tuning and dwell time calculation. (2) Full inventory of npm packages and Python dependencies pinned in requirements.txt and package-lock.json files across all repos, compared against PyPI and npm malware advisories published by CrowdStrike and Google during the Glassworm dismantlement — identifies any Glassworm-affiliated packages that may remain in use. (3) GitHub organization audit log export covering the campaign window, filtered for repository forks, collaborator additions, and Actions secret access — Glassworm's goal of downstream access means the hunt must extend to any repos that could have been used to distribute trojanized code to the organization's downstream users or customers. (4) MITRE ATT&CK navigator layer documenting all confirmed TTPs observed in this incident, mapped to detection gaps — drives prioritization of new Sigma rules and Sysmon configuration improvements.

Detection Guidance

Priority detection targets: (1) IDE processes (cursor.exe, windsurf.exe, positron.exe, VSCodium.exe, or their macOS/Linux equivalents) spawning unexpected child processes, particularly shell interpreters or network-connected scripts. (2) Outbound DNS queries or HTTP/S connections from IDE processes or package managers (npm, pip) to non-standard registries or recently registered domains. (3) Package install events referencing packages not in your approved inventory, cross-reference npm audit logs and pip install logs against CIS 2.1 baseline. (4) GitHub repository clone or pull events followed immediately by script execution on developer workstations. (5) New or modified files in IDE extension directories outside of normal update windows; apply File Integrity Monitoring (FIM) to these paths. (6) Service account or API token usage from developer workstations at unusual hours or accessing systems outside normal developer scope (NIST AU-2). NIST AU-6 (Audit Record Review and Analysis) and AU-12 (Audit Record Generation) should be validated to confirm logging is active for these sources. Note: Specific IOC values (IP addresses, domain names, package names, file hashes) are not confirmed from direct primary source review; validate against CrowdStrike's official technical reporting before operationalizing IOC-based detections.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`[not confirmed — see primary sources]`	C2 domains associated with Glassworm botnet infrastructure; four distinct C2 channels reported. Validate specific indicators against CrowdStrike blog and The Hacker News primary reporting before operationalizing.	LOW

Type	Value	Context	Confidence
HASH	`[not confirmed — see primary sources]`	Malicious IDE extension and trojanized package file hashes not confirmed from primary source review. Obtain from CrowdStrike advisory directly.	LOW

Framework Mappings

MITRE-ATTACK

- **T1071** — Application Layer Protocol
- **T1176** — Software Extensions
- **T1195.002** — Compromise Software Supply Chain
- **T1059** — Command and Scripting Interpreter
- **T1072** — Software Deployment Tools
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1568** — Dynamic Resolution

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **SI-7** — Software, Firmware, and Information Integrity
- **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
- **15.1** — Establish and Maintain an Inventory of Service Providers

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

SOC2-TSC

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1071	Application Layer Protocol	Command-And-Control
T1176	Software Extensions	Persistence
T1195.002	Compromise Software Supply Chain	Initial-Access
T1059	Command and Scripting Interpreter	Execution
T1072	Software Deployment Tools	Execution
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1568	Dynamic Resolution	Command-And-Control

Sources

Source	URL	Tier
GlassWorm Malware Takedown Disrupts Developer Supply Chain ...	https://thehackernews.com/2026/05/glassworm-malware-takedown-disrup...	T3
Inside CrowdStrike's Takedown of a Developer-Targeting Botnet	https://www.crowdstrike.com/en-us/blog/inside-crowdstrike-takedown-...	T3
CrowdStrike and Google have blocked 'Glassworm,' a botnet ...	https://gigazine.net/gsc_news/en/20260528-crowdstrike-takedown-glas...	T3
Glassworm Botnet Takedown Exposes Developer Supply Chain as	https://techjacksolutions.com/scc-intel/glassworm-botnet-takedown-e...	T3
Software Supply Chain Security: May 2026 Roundup | Cloudsmith	https://cloudsmith.com/blog/cloud-native-digest-may-2026	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 2026-05-30 19:07 UTC by TJS Security Command Center