INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-29 06:52 UTC

GlassWorm Campaign: Self-Propagating Malware Seeded via Open VSX VS Code Extension Marketplace

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0238
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	Visual Studio Code (VS Code), Open VSX Extension Marketplace (all versions accepting community-submitted extensions)
Published	2026-04-28T10:59:24
Discovery Source	Rss

Executive Summary

Threat actors are publishing malicious extensions to the Open VSX marketplace, a widely used alternative to Microsoft's official VS Code Extension Marketplace, as part of a campaign tracked as GlassWorm. The extensions carry self-propagating malware that, once installed on a developer workstation, can spread through CI/CD pipelines and into production systems and container images. Organizations with active software development teams using Open VSX face meaningful supply chain risk, including potential compromise of build artifacts and downstream customer-facing systems.

Technical Analysis

The GlassWorm campaign exploits insufficient submission controls in the Open VSX registry to publish trojanized VS Code extensions. The malware is classified under CWE-506 (Embedded Malicious Code), leveraging CWE-494 (Download of Code Without Integrity Check) and CWE-829 (Inclusion of Functionality from Untrusted Control Sphere) to describe the attack techniques used by the malware itself; these CWEs describe the *payload behavior*, not the Open VSX policy vulnerability that enabled publication. After installation, the malware propagates autonomously from developer workstations into connected build environments. MITRE ATT&CK techniques include T1176 (Browser Extensions, applied here to IDE extensions), T1554 (Compromise Software Supply Chain), T1195.001 (Supply Chain Compromise: Compromise Software Dependencies), T1072 (Software Deployment Tools), T1059 (Command and Scripting Interpreter), T1552.001 (Unsecured Credentials: Credentials in Files), and T1567 (Exfiltration Over Web Service). No CVE has been assigned to the Open VSX

submission bypass. This campaign is tracked from secondary threat reporting (Tier 3 news coverage from Dark Reading and The Hacker News); no Tier 1 or Tier 2 intelligence source has yet published official advisory. Human validation of source coverage is recommended before operational deployment of detections.

Action Checklist

1. Proactive Hardening: Audit all VS Code extensions installed across developer workstations and CI/CD build agents. Identify any extensions sourced from Open VSX (distinct from the official VS Code Marketplace). Evaluate whether Open VSX is a necessary tool in your environment given its lower verification bar compared to the official marketplace. If Open VSX extensions are in use, document business justification. Implement an approved extension allowlist enforced via policy (e.g., VS Code's extensions.allowedExtensionIds setting or equivalent MDM/GPO control). Require all developer tooling changes to go through a change management process.
2. Credential Hardening: Audit where credentials, API tokens, and cloud provider keys are stored. Remove any stored in VS Code settings, workspace files (.env, .vscode/settings.json), or other extension-accessible locations. Rotate credentials and tokens that may have been accessible from developer machines. Implement secure credential management (e.g., OS keychain, dedicated secret manager) for development workflows.
3. If Exposed, Containment: Suspend use of all extensions sourced from Open VSX in build pipelines pending verification. Isolate any developer systems where unvetted extensions are present. Search endpoint logs and EDR telemetry for anomalous process spawning from Code.exe or associated VS Code extension host processes. Review CI/CD pipeline logs for unexpected script execution or outbound network calls originating from build steps.
4. If Exposed, Detection: Audit installed extension manifests against a known-good baseline. Check for T1059-related indicators: unexpected interpreter invocations (cmd.exe, powershell.exe, bash) spawned by IDE processes. Check credential stores and environment variable files for unauthorized access consistent with T1552.001. Alert on anomalous behavior from VS Code extension host processes, particularly spawning unexpected child processes or making outbound connections to non-Microsoft endpoints.
5. If Exposed, Eradication: Remove all extensions not sourced from the official Microsoft VS Code Marketplace or not explicitly approved by your security team. Rebuild any CI/CD build agents or container images that may have been touched by infected developer environments. Rotate all credentials and tokens accessible from affected workstations, including git credentials, cloud provider keys, and API tokens.
6. If Exposed, Recovery: Rebuild developer workstations from clean images where possible, or reimage extension environments at minimum. Re-deploy CI/CD pipelines from verified clean source. Scan all container images built during the potential exposure window against known-good baselines before returning them to production. Monitor build pipeline outputs for unexpected artifacts or behavioral anomalies for at least 30 days post-remediation.
7. Post-Incident: Map findings against NIST SP 800-161 (Supply Chain Risk Management) and update your software supply chain risk register.

IR / Forensic Enrichment

Triage Priority	URGENT

Escalation Criteria	Escalate to CISO and legal/compliance immediately if forensic analysis confirms that rotated credentials (git tokens, cloud provider keys, or API tokens) were used to access production systems, customer data repositories, or regulated data stores (PII, PHI, PCI-scoped environments) during the exposure window, triggering breach notification obligations under applicable regulations (GDPR Article 33, HIPAA 45 CFR §164.410, or state data breach notification laws).
Recovery Notes	Before returning any rebuilt workstation or CI/CD agent to production, verify the integrity of the VS Code extension directory against a signed, version-controlled approved baseline and confirm no Open VSX registry endpoints are reachable from build agent network segments. All container images built during the exposure window must be rescanned with Trivy or equivalent and re-signed before production deployment; do not rely on previously cached image layers. Maintain elevated monitoring on build pipeline outputs, outbound network connections from developer endpoints, and cloud provider API activity logs for a minimum of 30 days post-remediation, given GlassWorm's demonstrated capability to persist through CI/CD artifacts that may not be immediately identified.
Forensic Artifacts	VS Code extension directory contents with preserved timestamps: `~/.vscode/extensions/` (Linux/macOS) or `%USERPROFILE%\.vscode\extensions\` (Windows) — GlassWorm-infected extensions will show anomalous activation event hooks in `package.json` (e.g., `onStartupFinished`) and may contain obfuscated JavaScript payloads in the extension's `out/` or `dist/` subdirectory not present in the legitimate Open VSX published source. | Sysmon Event ID 1 (Process Create) and Event ID 3 (Network Connection) logs filtered for parent processes matching `Code.exe` or `extensionHost` node instances — GlassWorm's self-propagation mechanism relies on spawning child interpreter processes (cmd.exe, powershell.exe, bash, sh) from the VS Code extension host to execute its propagation payload, leaving a distinctive parent-child chain in process telemetry. | CI/CD pipeline execution logs (GitHub Actions workflow run logs, Jenkins build console output, GitLab CI job traces) for all runs executed after the earliest possible extension installation date — the pipeline propagation vector means malicious code execution will appear as anomalous shell commands or unexpected outbound HTTP/S calls within build steps that previously had no such activity. | VS Code global storage and settings files: `%APPDATA%\Code\User\settings.json`, `%APPDATA%\Code\User\globalStorage\` (Windows) or `~/.config/Code/User/settings.json` and `~/.config/Code/User/globalStorage/` (Linux) — malicious extensions may persist harvested credentials, C2 configuration, or propagation scripts in the `globalStorage` namespace keyed to the extension's publisher ID. | Cloud provider and version control access logs covering the full exposure window: AWS CloudTrail `LookupEvents` filtered for the IAM principals whose credentials were stored on affected workstations, GitHub audit log filtered for the compromised personal access tokens, and Azure Activity Log or GCP Audit Log equivalents — these logs establish whether GlassWorm exfiltrated and weaponized credentials before rotation and define the true blast radius of the supply chain compromise.

Per-Action IR Details

Containment — Audit all VS Code extensions installed across developer workstations and CI/CD build agents. Identify any extensions sourced from Open VSX (distinct from the official VS Code Marketplace). Suspend use of Open VSX-sourced extensions in build pipelines pending verification. Isolate any developer systems where unvetted extensions are present.

NIST Phase: Containment

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

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

Compensating: On each developer workstation, dump installed VS Code extensions via: `code --list-extensions --show-versions > extensions_inventory.txt`. Cross-reference the publisher domain in each extension's package.json (located at `~/.vscode/extensions//package.json` on Linux/macOS or `%USERPROFILE%\.vscode\extensions\\package.json` on Windows) — Open VSX publishers will not match verified Microsoft Marketplace publisher IDs. For CI/CD build agents, grep Dockerfiles and pipeline YAML files (`.github/workflows/*.yml`, `.gitlab-ci.yml`, `Jenkinsfile`) for `code --install-extension` or `openvsx` references. Network-isolate flagged workstations via host firewall rule: `netsh advfirewall set allprofiles state on` (Windows) or `ufw default deny outgoing` (Linux) until cleared.

Evidence: Before isolating systems, capture: the full extension directory listing at `~/.vscode/extensions/` (Linux/macOS) or `%USERPROFILE%\.vscode\extensions\` (Windows) with file timestamps preserved (`ls -laR` or `dir /s /tc`); the VS Code extension host process tree via `Get-Process -Name 'Code' | Select-Object Id,Name,Path,StartTime` (Windows) or `ps auxf | grep -A5 extensionHost` (Linux); active outbound network connections from Code.exe or node.exe child processes via `netstat -anob` (Windows) or `ss -tp` (Linux) to capture any C2 beaconing before the network is severed; and a copy of each suspicious extension's `package.json` and any embedded `.vsix` contents, which may contain obfuscated activation event hooks targeting `onStartupFinished` or `workspaceContains` triggers used by GlassWorm to execute on IDE launch.

Detection — Search endpoint logs and EDR telemetry for anomalous process spawning from Code.exe or associated VS Code extension host processes. Review CI/CD pipeline logs for unexpected script execution or outbound network calls originating from build steps. Audit installed extension manifests against a known-good baseline. Check for T1059-related indicators: unexpected interpreter invocations (cmd.exe, powershell.exe, bash) spawned by IDE processes. Check credential stores and environment variable files for unauthorized access consistent with T1552.001.

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 IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Deploy Sysmon with a configuration capturing Event ID 1 (Process Create) and Event ID 3 (Network Connection), then query for processes where ParentImage matches `*\Code.exe` or `*\extensionHost.js` and ChildImage matches `*\cmd.exe`, `*\powershell.exe`, `*\bash.exe`, or `*\sh` — GlassWorm extensions spawn interpreters to execute embedded payloads on activation. Use this PowerShell one-liner to extract suspicious parent-child pairs from Windows Security Event Log (Event ID 4688, if process auditing is enabled): `Get-WinEvent -LogName Security | Where-Object {$_.Id -eq 4688} | Where-Object {$_.Message -match 'Code.exe'} | Format-List TimeCreated, Message`. For CI/CD, diff current pipeline execution logs against the last known-clean run using `git diff` on pipeline YAML and grep build logs for unexpected `curl`, `wget`, `Invoke-WebRequest`, or `nc` invocations. For T1552.001, check file access timestamps on `~/.gitconfig`, `~/.aws/credentials`, `%APPDATA%\Code\User\settings.json`, and `.env` files using `stat` (Linux) or `fsutil usn readjournal` (Windows) to identify reads by unexpected processes.

Evidence: Capture before remediation: Sysmon Event ID 1 logs showing process lineage rooted at `extensionHost` node processes (typically `node.exe` spawned by `Code.exe`); Windows Security Event ID 4663 (Object Access) for read attempts against credential files including `%APPDATA%\Code\User\settings.json`, `%USERPROFILE%\.gitconfig`, and `%USERPROFILE%\.aws\credentials`; CI/CD pipeline execution logs (GitHub Actions run logs, Jenkins build console output, GitLab job traces) for the window spanning Open VSX extension installation to present; network flow logs or Sysmon Event ID 3 records showing outbound connections from `node.exe` or `Code.exe` to non-Microsoft, non-Open-VSX IP ranges; and a memory dump of any live `extensionHost` process (`procdump -ma ` on Windows) if active malicious behavior is suspected, to recover in-memory payload that may not be persisted to disk.

Eradication — Remove all extensions not sourced from the official Microsoft VS Code Marketplace or not explicitly approved by your security team. Rebuild any CI/CD build agents or container images that may have

been touched by infected developer environments. Rotate credentials and tokens accessible from affected workstations, including git credentials, cloud provider keys, and API tokens stored in VS Code settings or environment files.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST IA-5 (Authenticator Management), NIST CM-7 (Least Functionality), CIS 5.2 (Use Unique Passwords), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Uninstall all non-approved extensions with: `code --uninstall-extension ` for each flagged entry from the inventory. For bulk removal, delete the entire `~/.vscode/extensions/` directory and reinstall only approved extensions from the Microsoft Marketplace using a controlled script. For CI/CD agents, retag compromised Docker images with a `QUARANTINE` label (`docker tag :quarantined-glassworm`) and remove them from rotation before rebuilding base images from a verified upstream digest (e.g., `docker pull node@sha256:`). Rotate all secrets found in `%APPDATA%\Code\User\settings.json`, `.env` files, `~/.gitconfig` (personal access tokens), `~/.aws/credentials`, and any secrets stored via VS Code's `SecretStorage` API (inspect via the extension's activation logs). Revoke and reissue tokens at the provider level — rotating the local file is insufficient if the token was already exfiltrated.

Evidence: Before eradication, preserve forensic copies of: the full contents of `%APPDATA%\Code\User\settings.json` and `globalStorage/` directory (which may contain extension-persisted data including harvested credentials or C2 configuration); a filesystem image or at minimum a file listing with hashes (`Get-FileHash -Algorithm SHA256` or `sha256sum`) of the `~/.vscode/extensions//` directory to support later YARA rule development; all CI/CD pipeline environment variable configurations (scrubbed copies) to document the credential exposure blast radius; and cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) covering the exposure window to determine whether rotated keys were used maliciously prior to rotation.

Recovery — Rebuild developer workstations from clean images where possible, or reimage extension environments at minimum. Re-deploy CI/CD pipelines from verified clean source. Scan all container images built during the potential exposure window against known-good baselines before returning them to production. Monitor build pipeline outputs for unexpected artifacts or behavioral anomalies for at least 30 days post-remediation.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST CP-10 (System Recovery and Reconstitution), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-2 (Baseline Configuration), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 7.3 (Perform Automated Operating System Patch Management)

Compensating: For container image integrity verification without a commercial scanner, use Trivy (free, open-source): `trivy image --exit-code 1 --severity HIGH,CRITICAL :` against all images built during the exposure window. Additionally, diff layer history of suspect images against clean baseline: `docker history --no-trunc ` to identify unexpected `RUN` instructions or added files introduced by a compromised build agent. For workstation rebuilds without MDM, use a version-controlled provisioning script (e.g., Ansible playbook or PowerShell DSC) that installs only approved VS Code extensions via `code --install-extension` from the Microsoft Marketplace, committed to a protected branch. Post-recovery, deploy a Sigma rule detecting `Code.exe` or `node.exe` spawning shells (sigma rule condition: `parentImage|endswith: 'Code.exe' AND image|endswith: ('cmd.exe','powershell.exe','bash')`) and run it against Sysmon logs daily for 30 days.

Evidence: Before returning systems to production, collect and retain: a clean-state baseline hash manifest of the rebuilt workstation's VS Code extension directory (`sha256sum ~/.vscode/extensions/**/*`) and CI/CD agent Docker image digests (`docker inspect --format='{{index .RepoDigests 0}}' `); build artifact SBOMs (Software Bill of Materials) for all container images produced post-recovery, generated via `syft -o spdx-json` for traceability; and a 30-day retention snapshot of CI/CD pipeline execution logs post-remediation to support anomaly comparison if GlassWorm indicators resurface through a missed infection vector.

Post-Incident — This campaign exposes a gap in extension sourcing controls. Implement an approved extension allowlist enforced via policy (e.g., VS Code's extensions.allowedExtensionIds setting or equivalent

MDM/GPO control). Require all developer tooling changes to go through a change management process. Evaluate whether Open VSX is a necessary tool in your environment given its lower verification bar compared to the official marketplace. Map findings against NIST SP 800-161 (Supply Chain Risk Management) and update your software supply chain risk register.

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), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Enforce the VS Code extension allowlist by deploying a `settings.json` template to all workstations via group policy preferences or a configuration management tool (Ansible, Puppet, Salt), setting `extensions.allowedExtensionIds` to an explicit array of approved Microsoft Marketplace extension IDs — this natively prevents VS Code from installing extensions outside the list without requiring commercial MDM. For supply chain risk register updates, create a structured entry documenting: Open VSX as a third-party software source, its lower publisher verification standard relative to the Microsoft Marketplace, GlassWorm as a known threat actor leveraging it, and the CI/CD propagation vector as the primary blast-radius multiplier. Schedule a quarterly review of all approved developer tooling sources against CISA's Known Exploited Vulnerabilities catalog and Open Source Security Foundation (OpenSSF) Scorecard ratings for extension publishers.

Evidence: For the post-incident review and lessons-learned documentation, preserve: the complete timeline of extension installation events reconstructed from VS Code extension host logs and filesystem timestamps; a full list of CI/CD pipeline runs, container image builds, and deployment events that occurred during the exposure window (to bound the potential supply chain contamination radius); any network IOCs (C2 domains, IP addresses, beacon intervals) identified during detection, formatted as structured threat intelligence in STIX 2.1 for sharing with your ISAC or peer organizations; and a copy of the original malicious extension's `package.json` and any recovered payload scripts, preserved as evidence and submitted to the Microsoft Security Response Center and Open VSX project maintainers to support takedown and improved publisher vetting.

Detection Guidance

Primary detection focus is on anomalous behavior from VS Code extension host processes. Key behavioral indicators: (1) VS Code extension host (extensionHost process) spawning unexpected child processes, particularly interpreters such as powershell.exe, cmd.exe, bash, or sh. (2) Outbound network connections from IDE processes to non-Microsoft, non-development endpoints. (3) File writes to CI/CD pipeline configuration files, Dockerfiles, or build scripts originating from developer workstation sessions. (4) Access to credential files or environment variable stores (.env files, ~/.gitconfig, ~/.aws/credentials) by extension-related processes, consistent with T1552.001. (5) Lateral movement from developer workstations into build infrastructure, particularly unexpected authentication events against CI/CD systems (Jenkins, GitHub Actions runners, GitLab CI agents). In SIEM, correlate process creation events (Sysmon Event ID 1) with parent process name containing 'extensionHost' or 'code'. Alert on any interpreter spawned by these parents. No confirmed IOCs have been published from verified sources at time of this report. Current source coverage is Tier 3 news reporting; treat any IOC from secondary sources with caution pending corroboration from official threat intelligence or CISA advisory.

Indicators of Compromise

Type	Value	Context	Confidence
DOMAIN	`open-vsx.org`	Open VSX marketplace domain — not inherently malicious, but the sourcing channel for campaign-delivered extensions. Use to audit extension provenance in developer environments.	LOW

Framework Mappings

MITRE-ATTACK

- **T1554** — Compromise Host Software Binary
- **T1176** — Software Extensions
- **T1552.001** — Credentials In Files
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1567** — Exfiltration Over Web Service
- **T1072** — Software Deployment Tools
- **T1059** — Command and Scripting Interpreter

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **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
- **15.1** — Establish and Maintain an Inventory of Service Providers

NIST-CSF-2

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

ISO-27001-2022

- **A.5.21** — Managing information security in the ICT supply chain

SOC2-TSC

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

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1554	Compromise Host Software Binary	Persistence
T1176	Software Extensions	Persistence
T1552.001	Credentials In Files	Credential-Access
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1567	Exfiltration Over Web Service	Exfiltration
T1072	Software Deployment Tools	Execution
T1059	Command and Scripting Interpreter	Execution

Sources

Source	URL	Tier
Security News	https://www.darkreading.com/application-security/fresh-glassworm-vs...	T3
Open VSX Bug Let Malicious VS Code Extensions Bypass ...	https://thehackernews.com/2026/03/open-vsx-bug-let-malicious-vs-cod...	T3
Malicious VS Code Extension Slipped Through Open VSX ...	https://www.reddit.com/r/CybersecurityClub/comments/1sgisdu/malicio...	T3
How we take down malicious Visual Studio Code extensions	https://checkmarx.com/zero-post/how-we-take-down-malicious-visual-s...	T3
■ Open VSX flaw lets attackers publish malicious VS Code ...	https://www.facebook.com/thehackernews/posts/-open-vsx-flaw-lets-at...	T3

Generated 2026-04-29 06:52 UTC by TJS Security Command Center