INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-28 06:33 UTC

GlassWorm Shifts to Deferred Payload Delivery: 73 OpenVSX Sleeper Extensions Target Developer Credentials

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0230
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	OpenVSX Registry, Visual Studio Code Marketplace, GitHub, npm ecosystem, macOS crypto wallet clients, developers using any of 73 identified malicious extensions
Published	2026-04-27T17:41:01
Discovery Source	Rss

Executive Summary

GlassWorm, a threat actor previously observed targeting developer environments, has re-emerged with 73 malicious extensions uploaded to the OpenVSX registry, clones of legitimate tools designed to appear benign during initial review. Six of those extensions have been confirmed actively delivering malware that steals developer credentials, secrets stored in local environments, and cryptocurrency wallet data on macOS. Organizations with developer teams using VS Code or Open VSX-compatible IDEs face direct risk of source code repository compromise, secret exposure, and potential supply-chain contamination.

Technical Analysis

GlassWorm uploaded 73 trojanized extensions to the OpenVSX registry, mimicking legitimate developer tools. The campaign's distinguishing tactic: initial uploads contained no malicious code, bypassing manual and automated review windows. Malicious payloads were delivered post-install via the extension update mechanism, exploiting the trusted update channel rather than the initial install artifact. Six extensions are confirmed malware-active. Targets include SSH private keys and other secrets stored in local developer environments (MITRE T1552.004), credentials stored in files (T1552.001), macOS cryptocurrency wallet clients (T1555), and OAuth/application tokens (T1528). The deferred delivery mechanism maps to CWE-494 (Download of Code Without Integrity Check) and CWE-829 (Inclusion of Functionality from Untrusted Control Sphere). Additional behaviors include obfuscated payloads (T1027, T1027.009), JavaScript execution (T1059.007), ingress tool transfer (T1105), and third-party software abuse for execution (T1072). Supply-chain

entry aligns with T1195.001. Certificate or signature validation bypass is suspected (CWE-295) but not confirmed. No CVE has been assigned. Trail of Bits vsix-audit is available as a static analysis tool for VSIX packages to identify suspicious extension behavior.

Action Checklist

1. Containment, Immediately disable or uninstall all 73 identified GlassWorm extensions from developer workstations. Treat any machine with a confirmed or suspected extension installed as compromised. Isolate affected systems from internal networks, CI/CD pipelines, and source code repositories pending investigation. Reference the extension list published via Truesec and HivePro advisories.
2. Detection, Audit all installed VS Code and Open VSX extensions across developer endpoints. Compare installed extension IDs and publisher names against the 73 identified malicious extensions. Review extension update logs for post-install payload downloads. Monitor for outbound connections from IDE processes to unexpected external hosts. Use Trail of Bits vsix-audit (github.com/trailofbits/vsix-audit) to scan VSIX packages for obfuscated or suspicious code. Look for JavaScript execution from extension host processes (T1059.007) and unexpected file reads targeting SSH key paths, .env files, or wallet data directories.
3. Eradication, Remove all confirmed and suspected malicious extensions. Rotate all developer secrets immediately: SSH keys, API keys, tokens, .env variables, cloud credentials, and any secrets that may have been stored or accessible in the local environment. Revoke and reissue OAuth tokens and application credentials (T1528 mitigation). For macOS developers, audit cryptocurrency wallet applications for unauthorized access or configuration changes.
4. Recovery, After secret rotation, validate that no compromised credentials have been used to access internal systems, cloud environments, or source code repositories. Review CI/CD pipeline configurations and recent commits for signs of injected malicious code (supply-chain persistence, T1195.001). Re-enable developer environments only after confirming clean extension sets. Enforce extension allowlisting via VS Code policy where supported.
5. Post-Incident, This campaign exposed gaps in extension vetting for deferred-payload delivery, initial code review does not catch post-install update abuse. Implement controls requiring extension integrity verification at update time (address CWE-494). Establish an approved extension allowlist for developer tooling. Evaluate IDE extension management policies under your software supply chain risk framework (NIST SP 800-161 or equivalent). Document the incident and update developer security awareness training to include IDE extension risks.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to CISO and legal counsel if forensic review confirms that stolen SSH keys, API tokens, or cloud credentials were used to access production systems, customer data repositories, or if any of the affected developers had write access to externally published npm packages — triggering potential supply-chain breach notification obligations and regulatory reporting timelines under applicable data protection law.

Recovery Notes	After credential rotation and CI/CD pipeline integrity verification, maintain elevated monitoring on all AWS CloudTrail, GitHub audit logs, and npm publish events for a minimum of 30 days using the newly rotated credential identities as watchlist entries, since GlassWorm's deferred-payload model means exfiltrated credentials may have been staged for delayed use. Re-enable developer environments only after vsix-audit clean-scans of the approved extension set are documented and the allowlist policy is enforced via MDM or GPO. Any anomalous API call, repository commit, or package publish from a previously affected developer identity during the 30-day watch window should be treated as an active incident and trigger immediate credential re-rotation.
Forensic Artifacts	~/.vscode/extensions//extension.js and compiled dist/ bundles — the deferred payload JavaScript downloaded post-install by GlassWorm's six active extensions will reside here; hash all files and compare against the clean VSIX published at install time to identify post-install modifications | macOS Unified System Log entries for 'Code Helper (Extension Host)' process — query with 'log show --predicate "process == \"Code Helper (Extension Host)\"" --last 72h' to reconstruct file access events against ~/.ssh/, .env files, and ~/Library/Application Support// during the extension active period | VS Code extension host output logs at ~/.config/Code/logs/ (Linux) or ~/Library/Application Support/Code/logs/ (macOS) — contain activation timestamps, uncaught exception stack traces from payload execution, and stdout/stderr from the malicious extension's Node.js runtime revealing C2 domains or exfiltration endpoints | DNS query history on developer workstations during the extension active window — on macOS via 'log show --predicate "subsystem == \"com.apple.mDNSResponder\""'; on Linux via systemd-resolved journal — identifies C2 infrastructure contacted by GlassWorm's deferred payload fetch mechanism distinct from legitimate VS Code telemetry domains (*.vscode.dev, *.visualstudio.com) | Git repository audit: output of 'git log --all --diff-filter=A --name-only --pretty=format:"" -- .github/workflows/* Jenkinsfile .circleci/config.yml' across all repos the affected developers had push access to — surfaces any CI/CD pipeline files added or modified during the compromise window as evidence of T1195.001 supply-chain persistence attempts

Per-Action IR Details

Containment — Immediately disable or uninstall all 73 identified GlassWorm extensions from developer workstations. Treat any machine with a confirmed or suspected extension installed as compromised. Isolate affected systems from internal networks, CI/CD pipelines, and source code repositories pending investigation. Reference the extension list published via Truesec and HivePro advisories.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-3 (Malicious Code Protection), CIS 2.3 (Address Unauthorized Software), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: On each suspect developer workstation, enumerate installed VS Code extensions via CLI: 'code --list-extensions' (Linux/macOS) or 'code.cmd --list-extensions' (Windows), then diff output against the 73 GlassWorm extension IDs from the Truesec/HivePro advisories. Uninstall matches with 'code --uninstall-extension <id>'. Immediately block outbound traffic from the VS Code extension host process (extensionHost.js) using host-based firewall rules (iptables on Linux, pf on macOS) targeting all non-RFC-1918 destinations on ports 80/443 until network forensics complete. For CI/CD isolation, revoke the developer's Personal Access Token on GitHub/GitLab before disconnecting from the network to prevent token-authenticated pull/push during the isolation window.

Evidence: Before uninstalling extensions, preserve: (1) the full contents of ~/.vscode/extensions/ (Linux/macOS) or %USERPROFILE%\.vscode\extensions\ (Windows) — each extension subdirectory contains the installed VSIX payload including any post-install-downloaded JavaScript; (2) the VS Code extension host process network connections captured via 'ss -tnp' (Linux) or 'netstat -b' (Windows) at time of isolation, noting any established

connections from 'extensionHost' to non-Microsoft, non-OpenVSX IP ranges; (3) macOS: unified log entries for the Code Helper (Extension Host) process via 'log collect --last 72h' targeting com.microsoft.VSCode.* subsystem before shutdown; (4) on Windows, preserve Prefetch files for Code.exe and node.exe (C:\Windows\Prefetch\) which record recently loaded DLLs and child process invocations by the extension host.

Detection — Audit all installed VS Code and Open VSX extensions across developer endpoints. Compare installed extension IDs and publisher names against the 73 identified malicious extensions. Review extension update logs for post-install payload downloads. Monitor for outbound connections from IDE processes to unexpected external hosts. Use Trail of Bits vsix-audit (github.com/trailofbits/vsix-audit) to scan VSIX packages for obfuscated or suspicious code. Look for JavaScript execution from extension host processes (T1059.007) and unexpected file reads targeting SSH key paths, .env files, or wallet data directories.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Deploy a one-shot osquery query across all developer endpoints: 'SELECT name, identifier, version, path FROM vscode_extensions;' (osquery 5.x with the VSCode extension table) to enumerate installed extensions fleet-wide without touching individual machines. For MITRE T1059.007 (JavaScript execution via extension host), use Sysmon Event ID 1 (Process Create) filtering on ParentImage matching '*extensionHost*' with child processes node.exe, cmd.exe, bash, or sh — deploy Sysmon config with the SwiftOnSecurity base template and add a ProcessCreate rule for ParentCommandLine containing 'extensionHost'. For file access detection targeting SSH keys and .env files, use Sysmon Event ID 11 (FileCreate) and Event ID 15 (FileCreateStreamHash) monitoring reads/writes to ~/.ssh/*, **/.env, ~/Library/Application Support/*/wallet* on macOS via auditd with a rule: '-a always,exit -F path=/Users/ -F name=.env -F perm=r -k glassworm_envread'. Run vsix-audit against every VSIX in ~/.vscode/extensions/ and pipe output through grep for 'eval', 'Buffer.from.*base64', 'http.get', 'child_process' to surface deferred payload download stubs.

Evidence: Preserve before analysis: (1) VS Code extension logs at ~/.config/Code/logs/ (Linux), ~/Library/Application Support/Code/logs/ (macOS), or %APPDATA%\Code\logs\ (Windows) — these contain extension activation events and uncaught exceptions from payload execution attempts; (2) npm/node HTTP request logs if NODE_DEBUG=http was set in the developer's environment — captures outbound GET/POST from extension host fetching deferred payloads; (3) macOS FSEvents or Linux inotify audit logs showing file reads against ~/.ssh/id_rsa, ~/.ssh/id_ed25519, **/.env, ~/Library/Application Support/Exodus/*, ~/Library/Application Support/Electrum/* (crypto wallet paths) within the extension host process context; (4) DNS query logs from the developer workstation (macOS: /var/log/system.log filtering 'mDNSResponder'; Linux: systemd-resolved query log via 'journalctl -u systemd-resolved') for domains contacted by the malicious extension's deferred payload delivery C2.

Eradication — Remove all confirmed and suspected malicious extensions. Rotate all developer secrets immediately: SSH keys, API keys, tokens, .env variables, cloud credentials, and any secrets that may have been stored or accessible in the local environment. Revoke and reissue OAuth tokens and application credentials (T1528 mitigation). For macOS developers, audit cryptocurrency wallet applications for unauthorized access or configuration changes.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST IA-4 (Identifier Management), NIST AC-2 (Account Management), NIST SI-2 (Flaw Remediation), CIS 5.3 (Disable Dormant Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Script secret rotation triage using a bash inventory pass before rotation: run 'grep -rn --include=".env" --include="*.env" --include="*.cfg" --include="*.conf" -E "(AWS_|GITHUB_TOKEN|NPM_TOKEN|API_KEY|SECRET)" ~/" to enumerate what secrets were present in files accessible to the extension host process. For SSH key compromise assessment, check ~/.ssh/known_hosts and ~/.ssh/authorized_keys for any entries added within the extension install window. Revoke GitHub PATs and OAuth apps via the GitHub API: 'gh auth token' to identify active tokens, then revoke via 'DELETE /applications/{client_id}/token' API call. For macOS crypto wallet triage, diff current Exodus wallet

config at ~/Library/Application Support/Exodus/exodus.wallet/ against any available backup — check file modification timestamps with 'ls -la --full-time' to identify unauthorized writes during the GlassWorm extension active period. Remove all 73 extension directories from ~/.vscode/extensions/ with 'rm -rf' rather than relying solely on 'code --uninstall-extension' to ensure no residual payload scripts remain.

Evidence: Preserve before eradication: (1) a forensic copy of ~/.vscode/extensions// directories in their entirety — the deferred payload JavaScript files will be present here if the second-stage download completed; (2) macOS Keychain access logs via 'log show --predicate "process == 'Code Helper'" --last 72h' to determine if the extension host accessed stored credentials from the system keychain; (3) shell history files (~/.bash_history, ~/.zsh_history) for any commands run during the suspected compromise window, particularly git operations, aws-cli calls, or npm publish commands that may indicate the attacker leveraged stolen credentials; (4) git credential helper cache contents ('git credential-cache' socket or ~/.git-credentials) which may have been read by the extension host; (5) npm token file at ~/.npmrc which stores registry authentication tokens frequently targeted by supply-chain credential theft campaigns.

Recovery — After secret rotation, validate that no compromised credentials have been used to access internal systems, cloud environments, or source code repositories. Review CI/CD pipeline configurations and recent commits for signs of injected malicious code (supply-chain persistence, T1195.001). Re-enable developer environments only after confirming clean extension sets. Enforce extension allowlisting via VS Code policy where supported.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST CM-7 (Least Functionality), NIST SI-7 (Software, Firmware, and Information Integrity), NIST AU-11 (Audit Record Retention), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: For CI/CD pipeline integrity verification without enterprise tooling: run 'git log --all --oneline --since=' on all repositories the affected developers had write access to, then 'git diff HEAD -- .github/workflows/ Jenkinsfile .circleci/config.yml' to inspect pipeline definition changes. For GitHub Actions specifically, check for new or modified workflow files and any new repository secrets added during the compromise window via the GitHub audit log API: 'GET /orgs/{org}/audit-log?phrase=action:repo.add_member+OR+action:secret.create'. Deploy VS Code extension allowlisting using a settings.json policy file deployed to %APPDATA%\Code\User\settings.json (Windows) or ~/Library/Application Support/Code/User/settings.json (macOS) with 'extensions.allowedExtensionIDs' array populated from your vetted allowlist — enforce via GPO (Windows) or MDM profile (macOS). Validate clean extension state post-recovery by re-running vsix-audit on newly installed extensions and confirming zero matches against the GlassWorm 73-extension IOC list before reconnecting developer machines to internal networks.

Evidence: Preserve before re-enabling systems: (1) GitHub/GitLab/Bitbucket audit logs for all API calls, repository clone/push events, and secret access events authenticated with credentials belonging to affected developers — request the full audit log export for the window from extension install date through rotation date; (2) cloud provider access logs (AWS CloudTrail, GCP Audit Logs, Azure Activity Log) filtering on IAM identity of affected developers for any ListBuckets, GetSecretValue, AssumeRole, or CreateAccessKey calls during the compromise window; (3) npm audit log from the registry for any package publish events under namespaces owned by affected developers — check npmjs.com publish history via 'npm info --json | jq .time'; (4) CI/CD build logs for pipeline runs triggered during the compromise window, specifically looking for unexpected npm install calls, curl/wget to external hosts, or base64-encoded payloads in build steps.

Post-Incident — This campaign exposed gaps in extension vetting for deferred-payload delivery — initial code review does not catch post-install update abuse. Implement controls requiring extension integrity verification at update time (address CWE-494). Establish an approved extension allowlist for developer tooling. Evaluate IDE extension management policies under your software supply chain risk framework (NIST SP 800-161 or equivalent). Document the incident and update developer security awareness training to include IDE extension risks.

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 SI-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Protection), NIST SI-2 (Flaw Remediation), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Address CWE-494 (Download of Code Without Integrity Check) at the extension update stage by scripting a pre-update integrity check: before any 'code --install-extension' or auto-update, use vsix-audit to hash-compare the incoming VSIX against a known-good baseline hash stored in a team-controlled registry (a simple git repo with a JSON manifest of approved extension IDs, versions, and SHA-256 hashes of the VSIX files). Implement a YARA rule scanning ~/.vscode/extensions/**/extension.js and **/dist/*.js for GlassWorm-specific deferred-payload patterns (e.g., 'setTimeout.*fetch', 'setInterval.*require("https")', base64-encoded URL strings) as a nightly cron job. For supply chain awareness training, create a tabletop scenario specifically modeling the GlassWorm deferred-payload technique — where an extension passes initial review but fetches malicious code on first activation after a delay — to close the gap in developer mental models about extension trust.

Evidence: Document for the lessons-learned record: (1) the timeline delta between extension publication on OpenVSX and first payload delivery activation — this measures the deferred-payload window and should inform the minimum observation period for new extension vetting; (2) a mapping of which of the 73 extensions were installed by how many developers and for how long — this defines the maximum credential exposure blast radius for breach notification assessment; (3) the specific JavaScript patterns vsix-audit flagged in the six confirmed active extensions — these become the YARA/Sigma detection signatures for the allowlisting enforcement system going forward; (4) any OpenVSX or VS Code Marketplace abuse reporting tickets filed and their response timelines — this informs future vendor escalation procedures under NIST IR-6 (Incident Reporting) for third-party registry compromise scenarios.

Detection Guidance

Primary detection focus is on extension inventory and post-install update behavior. Steps: (1) Run Trail of Bits vsix-audit against all installed VSIX packages to flag obfuscated or suspicious functionality. (2) Compare installed extension IDs and publisher metadata against the 73 GlassWorm extension identifiers published in Truesec and HivePro advisories. (3) Review VS Code extension host process logs for outbound network connections initiated after extension updates, particularly to non-Microsoft, non-Open VSX domains. (4) Search endpoint logs for file access to sensitive paths: ~/.ssh/, .env files, shell history, browser credential stores, and macOS Keychain or wallet application directories. (5) Monitor for JavaScript processes spawned from the VS Code extension host (extensionHost process) making network calls or reading credential files. (6) In CI/CD environments, review pipeline logs for unexpected secret access or code injection patterns in recent commits. Behavioral indicators: IDE extension making network calls to unfamiliar domains, unexpected reads of SSH private key files, .env files, or wallet data, and process execution chains originating from the VS Code extension host.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`https://open-vsx.org (73 malicious extensions — see Truesec/HivePro advisories for specific extension IDs)`	OpenVSX registry source of GlassWorm trojanized extensions	HIGH

Type	Value	Context	Confidence
URL	`See Truesec advisory (truesec.com/hub/blog/glassworm-self-propagating-vscode-extension) for confirmed extension identifiers`	GlassWorm campaign extension list — specific extension IDs not reproduced here to avoid unverified enumeration	HIGH

Framework Mappings

MITRE-ATTACK

- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1552.004** — Private Keys
- **T1552.001** — Credentials In Files
- **T1027.009** — Embedded Payloads
- **T1072** — Software Deployment Tools
- **T1528** — Steal Application Access Token
- **T1078.003** — Local Accounts
- **T1059.007** — JavaScript
- **T1204.002** — Malicious File
- **T1105** — Ingress Tool Transfer
- **T1027** — Obfuscated Files or Information
- **T1199** — Trusted Relationship
- **T1555** — Credentials from Password Stores

NIST-800-53R5

- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SC-8** — Transmission Confidentiality and Integrity
- **SC-17** — Public Key Infrastructure Certificates
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SC-13** — Cryptographic Protection

OWASP-TOP10-2021

- **A02:2021** — Cryptographic Failures
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **3.10** — Encrypt Sensitive Data in Transit
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **8.2** — Collect Audit Logs

HIPAA-SECURITY

- **164.312(d)** — Person or Entity Authentication

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures

NIST-CSF-2

- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1552.004	Private Keys	Credential-Access
T1552.001	Credentials In Files	Credential-Access
T1027.009	Embedded Payloads	Defense-Evasion
T1072	Software Deployment Tools	Execution
T1528	Steal Application Access Token	Credential-Access
T1078.003	Local Accounts	Defense-Evasion
T1059.007	JavaScript	Execution
T1204.002	Malicious File	Execution
T1105	Ingress Tool Transfer	Command-And-Control
T1027	Obfuscated Files or Information	Defense-Evasion
T1199	Trusted Relationship	Initial-Access
T1555	Credentials from Password Stores	Credential-Access

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/glassworm-malware-at...	T3
GlassWorm malware attacks return via 73 OpenVSX "sleeper ...	https://www.bleepingcomputer.com/news/security/glassworm-malware-at...	T3
trailofbits/vsix-audit: Security scanner for VS Code extensions - GitHub	https://github.com/trailofbits/vsix-audit	T3
GlassWorm - Self-Propagating VSCode Extension Worm - Truesec	https://www.truesec.com/hub/blog/glassworm-self-propagating-vscode-...	T3
Attackers Hijack Open VSX Extensions to Spread GlassWorm Malware	https://hivepro.com/threat-advisory/attackers-hijack-open-vsx-exten...	T3

Generated 2026-04-28 06:33 UTC by TJS Security Command Center