CSAI

cloud security alliance®

CSAI Foundation

Cloud Security Alliance AI Safety Initiative

GlassWorm Zig Dropper: Cross-IDE Developer Supply Chain Attack

Enterprise Implications of Multi-IDE Infection via Malicious Developer Extensions

Unofficial AI-assisted Research

2026-04-13

Key Takeaways

The GlassWorm campaign has extended its attack chain with a Zig-compiled native dropper — a materially different approach from prior waves that embedded payloads in Unicode-obfuscated JavaScript. This variant uses a platform-native binary that executes entirely outside the JavaScript runtime sandbox, acquiring full operating-system-level access on developer workstations [1]. Its distinguishing characteristic is lateral propagation: rather than targeting a single editor, the dropper systematically enumerates every IDE on the infected machine that supports VS Code-compatible extensions — including Cursor, Windsurf, and VSCodium — and silently installs a second-stage malicious extension into each one using each editor's own command-line tooling [1][2].

Security research published in 2024 [3], corroborated by subsequent reporting on this campaign [10], found that Zig-compiled payloads achieved EDR bypass under controlled testing conditions, suggesting that detection models may be less attuned to Zig's compiler toolchain fingerprints than to those of C or C++ equivalents. For enterprise security teams, this represents a meaningful detection gap: depending on the specific EDR configuration in use, a developer workstation may be compromised across its entire development toolchain before any endpoint alert is generated.

The downstream risk is materially elevated for organizations that have adopted AI-native coding tools, because those tools expand the adversary's leverage beyond credential theft to influence over AI-generated code artifacts. When Cursor, Windsurf, or similar AI-assisted editors are compromised at the extension layer, an adversary gains access not only to the developer's credentials but to the AI agent's operational context: the full repository the agent reads, the code the agent writes, and in many configurations the shell commands the agent executes. This represents a MAESTRO Layer 3 (Execution Environment) compromise [11] that can propagate silently into every artifact the developer produces after infection. Organizations should treat this campaign as a signal that developer workstation security demands the same structured governance applied to production systems.

Background

The GlassWorm Campaign in Context

GlassWorm is a sustained threat campaign first identified in March 2025, when Aikido Security researchers discovered malicious npm packages embedding JavaScript payloads within invisible Unicode variation-selector characters (Unicode ranges U+FE00–U+FE0F and U+E0100–U+E01EF) [4]. The technique caused the malicious code to render as blank space in virtually every editor and terminal, making it invisible to standard code review in most environments and impractical to detect manually at scale. The campaign expanded through the remainder of 2025 and into 2026, growing in both scale and technical sophistication across successive waves.

The campaign's largest single wave, documented in early March 2026, spread simultaneously across five hosting platforms. Researchers identified 72 malicious extensions on the Open VSX Registry, 88 npm packages attributed to a related variant, and more than 151 compromised GitHub repositories — across a campaign that has touched more than 400 software components in total since October 2025 [5][9]. A concurrent development in that wave was the abuse of VS Code extension manifest fields: rather than embedding malware directly in extension code, threat actors began exploiting the `extensionPack` and `extensionDependencies` manifest fields to establish transitive delivery chains, whereby a benign-appearing extension would be updated post-publication to list a malicious sibling as a dependency [5]. This technique allowed the campaign to leverage trust accumulated by extensions with established installation bases before the malicious dependency was introduced.

CSA published a research note covering that wave's transitive dependency technique and its implications for developer tooling governance [6]. The Zig dropper variant documented here represents the campaign's next evolutionary step, first reported publicly in April 2026 [1][2].

Structural Vulnerabilities in the Open VSX Ecosystem

The Open VSX Registry, maintained by the Eclipse Foundation, had documented scanning deficiencies that made it a structurally permissive environment for GlassWorm's extension-publishing operations. A security disclosure in late February 2026 revealed that Open VSX's pre-publish security scanning pipeline suffered from a logic error: scanner process failures returned the same value as "no scanner configured," causing the registry to treat scan failures as clean results and publish extensions without inspection [7]. The Eclipse Foundation announced mandatory pre-publish scanning following that disclosure [7]. Because AI-native IDEs such as Cursor and Windsurf source extensions from Open VSX rather than from Microsoft's proprietary Visual Studio Marketplace, this scanning deficiency

concentrated GlassWorm's exposure risk in precisely those environments. Prior to the February 2026 correction, GlassWorm's operators had an effectively open registration channel for extension publication in the registry serving a rapidly growing category of AI-assisted development tools.

The initial infection vector for the Zig dropper campaign was an extension named "specstudio.code-wakatime-activity-tracker," published on Open VSX as an apparent replica of WakaTime, a widely used developer productivity tool that measures time spent in the IDE [1]. WakaTime integrations are installed across a broad range of development environments, and they require only an API key to configure — a low-friction profile that makes the impersonation a plausible lure for developers who recognize the brand. The malicious extension reproduced sufficient functional behavior to avoid immediate suspicion while loading the Zig-compiled dropper before any legitimate WakaTime logic ran [1].

Security Analysis

The Zig Native Addon Technique

A Node.js native addon is a compiled shared library that loads directly into the V8 runtime process and executes at the operating system level, bypassing every security control that operates within the JavaScript engine's sandboxed execution model. The GlassWorm Zig dropper delivers two platform-specific native addons within the malicious extension's `./bin/` directory: `win.node` on Windows, packaged as a PE32+ DLL, and `mac.node` on macOS, compiled as a universal Mach-O binary supporting both x86_64 and ARM64 architectures [1]. When the extension activates, these addons load before any JavaScript executes — a sequence that is permitted by design in the VS Code extension runtime and enabled for any extension that ships a native addon in its package.

The decision to compile the dropper in Zig rather than C or C++ carries meaningful evasion advantages. Zig produces self-contained native binaries with minimal runtime dependencies and a distinct toolchain fingerprint that differs substantially from the patterns on which most commercial EDR behavioral models are trained. Research on the Zig Strike offensive toolkit, published in late 2024, demonstrated that Zig-compiled payloads could bypass several major EDR stacks including Microsoft Defender for Endpoint under testing conditions [3]. GlassWorm's use of Zig for this dropper stage follows the same logic: initial infection events are less likely to generate detection alerts, providing time for the second-stage payload to establish persistence before any remediation occurs.

The macOS binary retained debug symbols from the development environment, including a build path of `/Users/davidioasd/Downloads/vsx_installer_zig` [1]. While this may indicate an operational security oversight by the campaign operators, it does not materially reduce the technical effectiveness of the dropper.

Cross-IDE Propagation as a Multiplier

The dropper's core operational logic involves systematically enumerating the infected developer's workstation for every IDE that implements the VS Code extension API. The enumeration covers standard installation paths for VS Code stable and Insiders builds, Cursor, Windsurf, VSCodium, and Positron on both Windows (`%LOCALAPPDATA%` and `%ProgramFiles%` directories) and macOS (`/Applications/`) [1]. For each discovered IDE, the dropper fetches a second-stage `.vsix` installer from an attacker-controlled GitHub repository and invokes the target IDE's own command-line extension installer to perform a silent installation [1]. Because the installer is each editor's own tooling, the installation process does not trigger elevated-privilege prompts or system security dialogs.

This propagation pattern has an important organizational implication: containment of the infection requires identifying and remediating every IDE installed on the affected workstation, not merely the one through which initial compromise occurred. A developer who had WakaTime installed in multiple editors would have all of those editors compromised from the first activation of the malicious extension. The second-stage extension, named "floktokbok.autoimport," impersonates the legitimate "steoates.autoimport" extension, which has more than five million installs on the Visual Studio Marketplace [1][2]. The impersonation relies on typographic similarity and publisher-name obscurity rather than namespace collision, making visual inspection of an installed extension list an unreliable detection method.

Payload Architecture and C2 Resilience

Once installed across all discovered IDEs, the second-stage extension implements a multi-capability infostealer and persistence framework. Its first action is to check whether the system is configured for a Russian locale; if it is, execution terminates without further activity [1][4]. This geofencing behavior is consistent across all documented GlassWorm waves and is indicative of an operator seeking to limit exposure to domestic law enforcement action or simply to avoid compromising systems in the operators' own region — a pattern common among financially motivated threat actors.

On non-geofenced systems, the extension resolves its active command-and-control server by querying a Solana blockchain transaction memo [4]. This architecture, documented first in the March 2026 GlassWorm wave, is inherently resilient against network-layer interdiction: because the Solana blockchain

is a distributed, public ledger with no central chokepoint, there is no domain to sinkhole, no IP address to block at the network perimeter, and no hosting provider to compel into takedown. The threat actor can rotate the active C2 address by submitting a new Solana transaction memo, restoring operational capability without any infrastructure change visible to defenders. Conventional indicators-of-compromise (IoC) feeds that distribute IP address blocklists provide limited value for disrupting GlassWorm's C2 communication layer, though they remain useful for detecting other threat actors operating on the same network.

From the resolved C2 endpoint, the framework downloads and installs a remote access trojan (RAT) that deploys a malicious browser extension across Chrome, Edge, Brave, Opera, and other Chromium-based browsers, as well as Firefox [4]. The browser extension captures keystrokes, extracts session cookies and authentication tokens, takes periodic screenshots, and receives operator commands over a WebSocket channel. It actively targets authentication tokens for GitHub, npm, and Open VSX publisher accounts — precisely the credentials that, once stolen, enable the campaign to maintain persistence in the extension registry ecosystem by force-pushing poisoned updates from legitimate maintainer accounts [4][8]. The framework also targets browser extensions associated with a range of cryptocurrency wallet providers [4].

The AI-Native IDE Amplification Risk

The industries most exposed to GlassWorm's cross-IDE propagation are those that have adopted AI-native coding tools at scale. Cursor and Windsurf, both explicitly targeted by the Zig dropper, are designed around the premise that the AI agent integrated into the IDE should have broad access to the developer's working environment: read access to the full repository, write access for code generation and editing, and in common configurations the ability to execute shell commands and run tests directly from within the IDE interface.

An adversary who compromises such an environment through an infected extension may gain a persistent vantage point from which to observe and potentially influence AI-generated code artifacts the developer produces, depending on the specific extension API surface the IDE exposes to extension code. Malicious code can be quietly injected into pull request submissions. AI-generated package imports can be subtly redirected toward attacker-controlled dependencies. Shell command suggestions from the AI agent can be manipulated if the extension has access to the AI completion pipeline. None of these post-compromise manipulations requires any additional network connection or privilege escalation beyond the initial IDE extension installation, because the AI agent itself operates with the developer's full permissions.

This attack surface is captured by MAESTRO's Layer 3 taxonomy (Execution Environment) [11], which addresses threats to the environments within which AI agents operate. A compromised execution environment subverts all agent outputs without requiring any compromise of the underlying model, making it a particularly high-leverage attack vector for adversaries targeting organizations that rely on AI-assisted development. The infection of multiple IDEs simultaneously through the GlassWorm dropper means that a developer who attempts to remediate by switching editors remains under the adversary's control.

Recommendations

Immediate Actions

Organizations should immediately cross-reference their developer endpoint inventory against two specific indicators of compromise: the extension identifier "specstudio.code-wakatime-activity-tracker" in any IDE extension list, and the extension identifier "floktokbok.autoimport" in any IDE extension list [1]. Both extensions should be treated as confirmation of full workstation compromise. The file hashes `2819ea44e22b9c47049e86894e544f3fd0de1d8afc7b545314bd3bc718bf2e02` (win.node) and `112d1b33dd9b0244525f51e59e6a79ac5ae452bf6e98c310e7b4fa7902e4db44` (mac.node) can be used to scan local file systems for the Zig dropper binary through endpoint management tooling [1].

Any developer whose machine is identified as affected should immediately revoke and rotate all credentials accessible from that system. Priority rotation targets include GitHub personal access tokens, npm publishing tokens, cloud provider access keys, SSH keys, and any Open VSX or other extension registry publisher credentials. All repositories and packages published from the compromised account should be audited for unauthorized commits, version bumps, or dependency changes. Given that the Zig dropper variant was first reported in April 2026 [1], a lookback of at least 90 days from the date of detection is recommended as a conservative margin; organizations with evidence of earlier credential exposure may need to extend this window. Organizations using the open-source glassworm-hunter scanner, maintained by AFINE, can run it against local IDE extension directories, npm package caches, and Python environments to identify additional indicators before re-imaging [9].

Affected workstations should be re-imaged rather than remediated in place. Because the payload's full persistence mechanisms have not been characterized in public reporting, extension-level remediation on a live system may carry a risk of incomplete cleanup. Out of an abundance of caution, re-imaging is the

recommended approach over attempting in-place remediation.

Short-Term Mitigations

Organizations should establish formal governance over IDE extensions as part of developer endpoint policy. This means maintaining an approved extension list, requiring review before any new extension is installed, and defaulting to the official Visual Studio Marketplace over Open VSX where functional equivalents are available — given the documented difference in vetting rigor between the two registries that persisted through early 2026. For developers who require Open VSX extensions, organizations should verify that the Eclipse Foundation's post-disclosure mandatory pre-publish scanning has been applied to the specific extension version before installation [7].

Endpoint detection configurations should be tuned to alert on the behavioral patterns characteristic of the GlassWorm dropper: specifically, the creation of `.node` files in IDE extension subdirectories, IDE command-line processes spawning child processes that write to temporary directories, and those same child processes invoking additional IDE CLI installers. These behavioral patterns appear during the dropper's cross-IDE propagation step and are not characteristic of normal extension operation. Organizations with browser enterprise management capabilities should enforce extension allowlists across Chromium-based browsers and Firefox to prevent force-installation of unauthorized browser extensions even when an adversary has code-execution access on the workstation.

Blocking Solana RPC endpoints broadly may introduce false positives in organizations with legitimate Web3 tooling or developer workflows that use the Solana ecosystem; evaluate the tradeoff based on your environment's actual Solana usage profile. Behavioral endpoint detection is the appropriate primary control at the C2 communication layer for this campaign.

Strategic Considerations

The GlassWorm campaign's cross-IDE propagation technique is a structural indicator that developer workstation security has become a first-order supply chain risk. An organization that has invested substantially in securing its CI/CD pipelines, artifact registries, and container images while treating developer laptops as trusted internal endpoints has a material gap that campaigns like GlassWorm are designed to exploit. Developer workstations should be brought under continuous endpoint detection, should enforce secrets management policies that prevent plaintext credential storage, and should be provisioned with least-privilege access to production systems even where that creates short-term operational friction.

The adoption of AI-native development tools accelerates the urgency of this posture shift. Unlike traditional IDEs, AI coding assistants are designed to operate as autonomous agents within the developer's environment, and that autonomy translates directly into adversarial leverage when the agent's execution environment is compromised. Organizations deploying AI-assisted development tools should conduct threat model reviews using the MAESTRO framework [11] with IDE extension compromise explicitly included as an initial access scenario, and should evaluate whether their AI tooling vendors provide any isolation mechanisms between extension code and the AI agent's core execution pipeline.

Extension registries — including corporate or project-internal registries — should adopt pre-publication security controls analogous to those now mandated at Open VSX. Organizations that maintain private extension registries for internal tooling distribution should implement automated scanning of extension packages at publication time, require code signing for extension artifacts, and publish a software bill of materials (SBOM) for each extension version to support future incident investigation. These controls close the structural gap that GlassWorm has exploited across every wave of its campaign: the absence of consistent vetting between submission and developer installation.

CSA Resource Alignment

AI Controls Matrix (AICM): GlassWorm's targeting of AI-native coding environments maps directly to AICM's Application Provider control category, which addresses security requirements for systems that operate AI agents on behalf of end users or organizations. The campaign demonstrates that the AI application security perimeter includes the IDE extension ecosystem through which AI coding agents are configured and extended. Organizations applying AICM Application Provider controls to their AI deployments should extend that scope to include developer workstations running AI-assisted editors.

Cloud Controls Matrix (CCM): The Supply Chain Management and Transparency domain (STA-09 through STA-12) addresses third-party component vetting, software bill of materials requirements, and supplier risk management. GlassWorm's exploitation of registry vetting gaps represents a failure at the STA layer of developer tooling supply chains. Organizations applying CCM STA controls to their cloud and software supply chains should assess whether equivalent rigor is applied to IDE extensions and package managers used in their development workflows.

Software Transparency: Securing the Digital Supply Chain: CSA's guidance on software supply chain security addresses CI/CD pipeline integrity, open-source software risk management, and SBOM requirements in terms directly applicable to the extension registry and package ecosystem vectors

GlassWorm has exploited. That guidance's recommendations on pre-publication scanning, artifact signing, and continuous monitoring of third-party components apply at every layer of the developer tooling supply chain, including IDE extensions.

MAESTRO (Multi-Agent Environment Security Threat Reasoning and Operational Framework) [11]: The compromise of AI-native IDEs through malicious extensions constitutes a MAESTRO Layer 3 (Execution Environment) attack. By subverting the environment within which the AI coding agent operates, the adversary gains the ability to influence all agent outputs — including generated code, dependency suggestions, and executed shell commands — without requiring any access to the underlying model. Organizations conducting MAESTRO threat model exercises for their AI-assisted development workflows should include IDE extension supply chain compromise as an explicit threat scenario and evaluate whether any isolation boundary exists between extension execution and agent pipeline access.

Zero Trust Guidance: CSA's Zero Trust guidance — verify explicitly, use least privilege, and assume breach — applies directly to the question of developer workstation access controls. The GlassWorm RAT's credential harvesting capabilities demonstrate that a compromised developer machine should be treated as an untrusted network participant capable of exfiltrating all credentials accessible from it. Zero Trust network access controls applied to developer workstations should enforce continuous session verification and should limit the scope of internal resources accessible from any single workstation, regardless of its apparent identity posture.

References

[1] Aikido Security. "GlassWorm goes native: New Zig dropper infects every IDE on your machine." Aikido Security Blog, April 2026.

[2] The Hacker News. "GlassWorm Campaign Uses Zig Dropper to Infect Multiple Developer IDEs." The Hacker News, April 2026.

[3] KPMG Netherlands. "Zig Strike: The ultimate toolkit for payload creation and evasion." KPMG Insights, December 2024.

[4] The Hacker News. "GlassWorm Malware Uses Solana Dead Drops to Deliver RAT and Steal Browser, Crypto Data." The Hacker News, March 2026.

[5] The Hacker News. "GlassWorm Supply-Chain Attack Abuses 72 Open VSX Extensions to Target Developers." The Hacker News, March 2026.

[6] Cloud Security Alliance. "GlassWorm: Open VSX Transitive Dependency Supply-Chain Escalation." CSA AI Safety Initiative, March 2026.

[7] The Hacker News. "Open VSX Bug Let Malicious VS Code Extensions Bypass Pre-Publish Security Checks." The Hacker News, March 2026.

[8] The Hacker News. "GlassWorm Attack Uses Stolen GitHub Tokens to Force-Push Malware Into Python Repos." The Hacker News, March 2026.

[9] AFINE. "GlassWorm Hunter – Detect GlassWorm Supply Chain Attacks." AFINE Blog, 2026.

[10] Security Affairs. "GlassWorm evolves with Zig dropper to infect multiple developer tools." Security Affairs, April 2026.

[11] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance Blog, February 2025.

