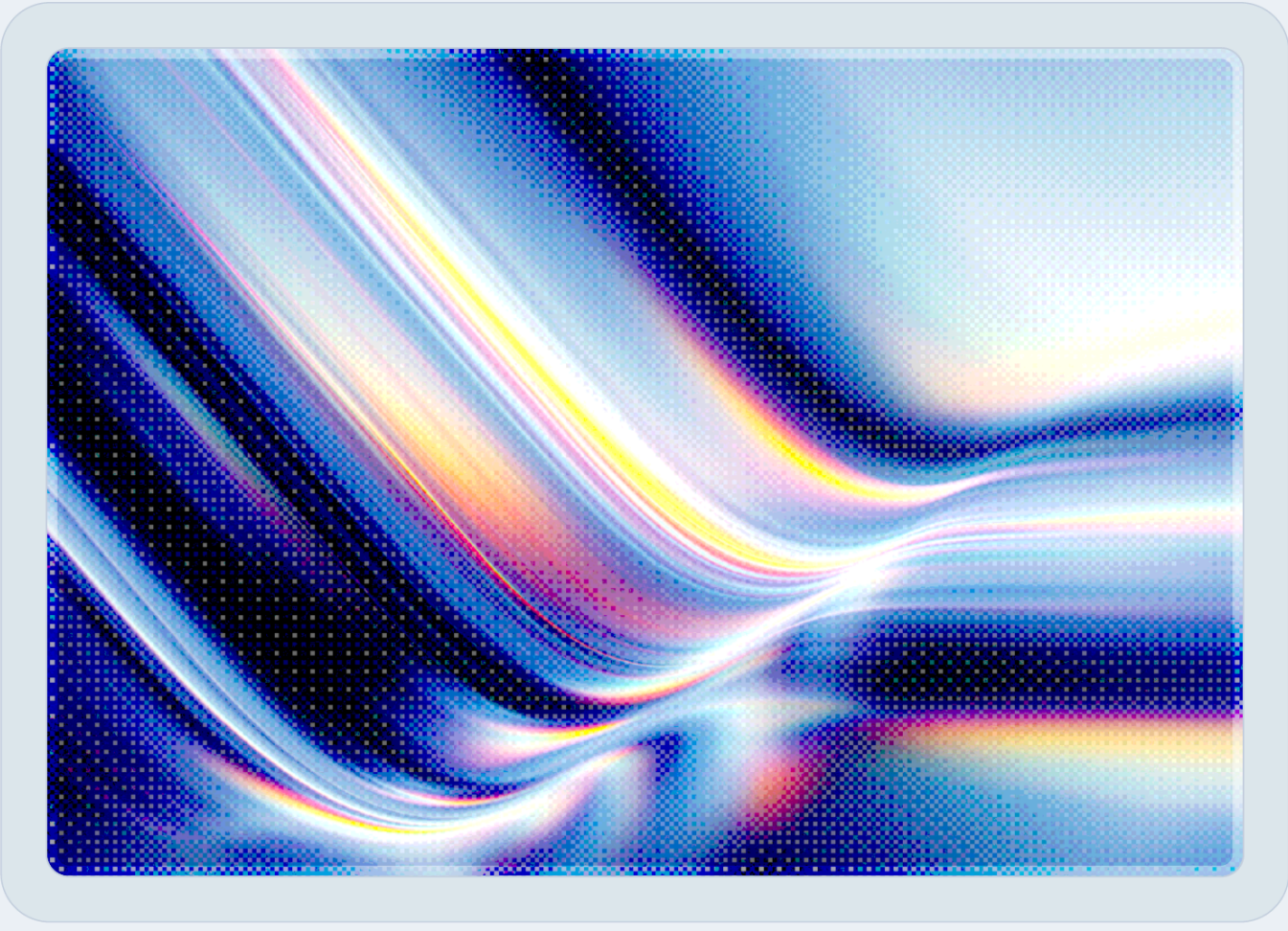
CSAI Foundation | Cloud Security Alliance

Developer Ecosystems as Critical Infrastructure

GlassWorm, TeamPCP, and the Cascading Supply Chain Risk Model

2026-05-27

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Table of Contents

Executive Summary

The software supply chain has become one of the most consequential attack surfaces in modern computing. Where earlier campaigns exploited specific vulnerabilities in shipping code, the threat landscape of 2025–2026 reveals a more troubling evolution: adversaries are not targeting individual applications but are instead systematically compromising the infrastructure through which all software is built, distributed, and maintained. When the tools developers use to write, scan, and deploy code are themselves weaponized, every downstream artifact becomes suspect — and the scope of organizational exposure expands from a single compromised application to the entirety of that organization's software-producing capability.

Two campaigns that crystallized this strategic shift are GlassWorm and TeamPCP. GlassWorm, first identified in October 2025, introduced a novel attack paradigm to the developer tooling space: a self-propagating worm that spreads through VS Code and OpenVSX extension ecosystems, harvesting credentials and establishing persistent access via blockchain-anchored command-and-control infrastructure. TeamPCP, active since at least late 2025, has demonstrated a methodical campaign of poisoning widely trusted security and AI tooling — scanners, gateways, and package management utilities — while simultaneously developing self-propagating capabilities under a sub-campaign designated Mini Shai-Hulud. In one documented instance, TeamPCP's tooling defeated SLSA Build Level 3 provenance attestations, a control that many enterprises treat as a trust anchor for their supply chain integrity programs.

These campaigns point to a systemic vulnerability in how the industry has conceptualized supply chain risk. The prevailing model treats supply chain attacks as discrete events: a compromised package here, a poisoned dependency there. The observed attack patterns suggest a more interconnected threat architecture in which compromise in one layer of the developer ecosystem enables automated, cascading propagation into adjacent layers. This paper formalizes that observation as the Cascading Supply Chain Risk Model and examines its implications for enterprise security architecture.

The document concludes with recommendations organized across immediate, short-term, and strategic timeframes, and maps its findings to CSA's MAESTRO framework, the AI Controls Matrix (AICM), DevSecOps guidance, and Zero Trust principles.

Introduction: The Developer Ecosystem Under Siege

Modern software development rests on a layered ecosystem of trust relationships. Developers install extensions into their integrated development environments without reviewing the underlying source code. Build systems fetch hundreds of transitive dependencies automatically. CI/CD pipelines execute arbitrary

scripts with elevated permissions. Security tooling downloaded from public registries scans production artifacts. Each layer in this stack assumes that its inputs are trustworthy — and each assumption represents a potential attack surface.

This trust architecture was not engineered for adversarial exploitation at scale. It emerged organically from the economics of open-source collaboration: sharing code is efficient, package registries reduce duplication, and automated pipelines accelerate delivery. The same properties that make developer tooling ecosystems productive — ubiquity, automatic installation, deeply embedded trust relationships, and high-privilege execution — make them extraordinarily attractive to threat actors seeking leverage over many downstream targets simultaneously.

Security researchers and government agencies have increasingly characterized open-source registries and developer infrastructure as components of critical national infrastructure [1][2]. The 2026 State of the Software Supply Chain report published by Sonatype noted that open-source consumption continues to accelerate, with modern applications often consisting of 80–90 percent open-source components [3]. CISA's 2025 Year in Review identified software supply chain integrity as a persistent top priority, and in March 2026, NSA and seven allied nations released joint guidance specifically addressing supply chain risks in AI and machine learning systems [4]. The policy community has arrived at a conclusion that the attack campaigns of 2025–2026 reinforce empirically: developer ecosystems are infrastructure, and they are under sustained, sophisticated attack.

This paper examines two representative campaigns that illustrate distinct but complementary threat models, then synthesizes their observed behaviors into a unified analytical framework — the Cascading Supply Chain Risk Model — that can guide enterprise risk assessment and defensive investment.

GlassWorm: The Self-Propagating Developer Worm

Discovery and Initial Scope

GlassWorm was first disclosed publicly in October 2025 by researchers at Koi Security, who identified seven malicious extensions in the OpenVSX marketplace — the primary extension registry for VS Code-compatible editors — that had collectively accumulated approximately 35,800 downloads [6]. The discovery was notable not merely for the presence of malicious code in a popular extension registry, a threat category that had been documented in prior years, but for the mechanism by which the malware sustained and expanded its operations.

The campaign returned to prominence in March 2026 when a second major wave was documented. By that point, researchers had attributed more than 72 malicious extensions on OpenVSX and over 151 compromised GitHub repositories to the GlassWorm infrastructure [7]. A third wave documented in March 2026 extended the campaign's reach to npm packages and additional VS Code extensions, bringing the reported total of compromised components to more than 433 across GitHub, npm, and the VS Code marketplace [8].

Technical Architecture

GlassWorm's technical design departs significantly from earlier developer-targeted malware in several important respects. The malware employs invisible Unicode characters — zero-width joiners, zero-width non-joiners, and other non-printing codepoints — to embed malicious logic within code that appears clean in standard code editors and diff views [6][9]. This obfuscation technique is particularly effective in developer environments because the tooling developers rely upon to review code — their IDEs, code review interfaces, and terminal output — is precisely the same tooling the attacker has already compromised.

The command-and-control architecture relies on the Solana blockchain as a primary channel, with Google Calendar serving as a backup mechanism [6][7]. The choice of decentralized infrastructure for C2 is strategically significant: blockchain-anchored communications are substantially more resilient to takedown than traditional C2 domains or IP addresses. When defenders notify hosting providers or request domain seizures, blockchain records remain immutable. This design reflects adversarial awareness that rapid C2 disruption is a standard defensive response to malware campaigns.

Once installed, GlassWorm pursues a broad credential-harvesting mandate. Documented targets include GitHub authentication tokens, VS Code credential stores, npm publish tokens, cryptocurrency wallets across 49 documented extensions, SSH private keys, and general developer environment configuration data [7]. The malware also includes a full remote access trojan (RAT) capability, enabling persistent post-compromise access to infected developer workstations [6]. The self-propagating component enables the malware to spread from one compromised developer environment to others through the developer's existing access to repositories and publishing credentials, achieving worm-like expansion through the very trust relationships that define the open-source ecosystem.

Attribution Indicators

Attribution analysis by researchers identified indicators suggesting the malware's operators are Russian-speaking. GlassWorm includes a locale check that suppresses execution when the detected system locale corresponds to Russian-language settings [7]. This technique — skipping execution on Russian-language

systems — is a documented pattern in financially motivated and state-adjacent Russian-speaking threat operations, serving as a crude but common geofencing mechanism. CSA notes that attribution inferences of this kind are probabilistic and that actors may deliberately employ such markers to misdirect attribution.

TeamPCP: Coordinated Supply Chain Poisoning at Scale

Threat Actor Profile

TeamPCP is a financially motivated threat actor first tracked by security researchers in late 2025 under multiple aliases, including DeadCatx3, PCPcat, ShellForce, and CipherForce [10][11]. Unlike GlassWorm, which primarily targets individual developer workstations and credential stores, TeamPCP's campaign demonstrates a more systematic focus on poisoning widely trusted security and infrastructure tooling — components whose compromise has an outsized effect because they sit in the verification and scanning layer of many organizations' software development lifecycles.

The actor's targeting logic reflects a sophisticated understanding of the software supply chain's trust topology. Security scanners and CI/CD tooling are often explicitly exempted from the scrutiny they apply to other software: enterprises generally do not scan their scanners, sign their signing tooling, or apply the same rigor to the security of their security tools as they apply to the security of production applications. TeamPCP exploits this blind spot systematically.

Campaign Scope and Notable Compromises

Across documented incidents spanning late 2025 through May 2026, TeamPCP compromised a notable set of widely deployed open-source security and infrastructure components. The table below summarizes confirmed compromise events attributed to the actor:

Component	Category	Reported Compromise Date	Significance
Aqua Security Trivy	Container vulnerability scanner	March 2026	Scans container images in CI/CD pipelines across thousands of organizations
KICS (Checkmarx)	Infrastructure-as-code scanner	March 2026	Analyzes IaC configurations before deployment

Component	Category	Reported Compromise Date	Significance
LiteLLM AI Gateway	AI/LLM proxy and routing layer	March 2026	Routes AI model calls in enterprise AI deployments
Bitwarden CLI npm package	Credential manager CLI	April 2026	Used in CI/CD workflows for secrets retrieval
Checkmarx Jenkins AST Plugin	Static analysis pipeline integration	May 2026	Embedded in Jenkins-based CI/CD pipelines
TanStack npm packages	UI component library ecosystem	May 2026	Broadly used in JavaScript/TypeScript applications

Sources: [10][11][12][13][14]

The April 21–23, 2026 period saw a coordinated three-wave attack by TeamPCP-affiliated actors targeting npm, PyPI, and Docker Hub simultaneously, with documented exfiltration of API keys, cloud credentials, and CI/CD secrets [14]. The coordination of registry attacks across multiple ecosystems within a 48-hour window represents a tactical evolution from earlier supply chain campaigns, which typically targeted one registry at a time.

Mini Shai-Hulud: The Self-Propagating Sub-Campaign

The most technically advanced element of TeamPCP's operational portfolio is a sub-campaign that researchers designated Mini Shai-Hulud, active from approximately September 2025 through at least May 2026 [15][16]. The campaign's most significant reported achievement is the defeat of SLSA Build Level 3 provenance attestations.

SLSA (Supply-chain Levels for Software Artifacts) is a framework developed collaboratively by Google and the broader security community to establish verifiable provenance for software artifacts. Build Level 3 — the highest defined level in the framework at time of writing — requires that builds occur in an isolated, hardened build environment and that cryptographic attestations link every published artifact to a specific, auditable build process [17]. The SLSA framework has been broadly recommended as a supply chain integrity control by CISA, NIST, and the Linux Foundation, and many enterprises treat SLSA Level 3 attestations as a meaningful trust signal when evaluating third-party packages.

Mini Shai-Hulud reportedly compromised SLSA Level 3 attestations by targeting the CI/CD runner environments in which attested builds occur, rather than attempting to forge attestations directly [15]. By compromising secrets accessible within the build environment — including the signing credentials used to produce provenance attestations — the campaign could produce cryptographically valid attestations for maliciously modified artifacts. Researchers documented the campaign affecting 502 unique packages across 1,055 published versions, with the majority of affected packages in the npm ecosystem [15][16]. In one documented incident, malicious artifacts were published across 42 TanStack packages within approximately six minutes of initial access, demonstrating the speed advantage that automated tooling provides to supply chain attackers [16].

The Cascading Supply Chain Risk Model

Motivation for a New Analytical Framework

The prevailing approach to software supply chain risk treats each incident as a bounded event: a specific package is compromised, a specific vulnerability is disclosed, a specific dependency is affected. Remediation follows the same logic: identify affected versions, update dependencies, rotate exposed credentials. This event-centric model was adequate when supply chain attacks were relatively rare and technically unsophisticated. It is increasingly inadequate when facing threat actors whose capabilities include self-propagating malware, automated multi-registry attacks, and systematic defeat of the controls enterprises have built to detect supply chain compromises.

GlassWorm and TeamPCP together illustrate a different threat architecture — one in which compromise in one layer of the developer ecosystem automatically enables or accelerates compromise in adjacent layers. The Cascading Supply Chain Risk Model formalizes this observation as an analytical framework.

Model Description

The Cascading Supply Chain Risk Model characterizes software supply chain risk as a function of three interacting properties: **node criticality**, **propagation vectors**, and **cascade depth**. These properties are drawn from established resilience theory applied to networked infrastructure systems [18][19] and adapted to the specific topology of software development ecosystems.

Node criticality describes the downstream impact multiplier of a given supply chain component. A component used by millions of projects, or one that sits in a verification or signing role for other components, has higher criticality than one used narrowly. By this measure, security scanners, CI/CD

infrastructure, package manager clients, and IDE tooling have extremely high node criticality regardless of their own attack surface area — because their compromise does not merely affect the organization using them directly, but potentially affects every artifact they process or every project that installs them.

Propagation vectors describe the mechanisms by which an initial compromise can spread to adjacent nodes. GlassWorm's propagation vector is primarily credential-based: once installed in a developer's IDE environment, it harvests publishing tokens that can be used to inject malicious code into packages the developer maintains. TeamPCP's CI/CD-focused propagation follows a different path: compromising a scanner installed in a CI/CD pipeline may provide access to the environment variables and secrets present in that pipeline, which may in turn include cloud credentials, container registry credentials, or keys for signing other artifacts.

Cascade depth describes how many transitional steps can occur between an initial compromise and the downstream effects that affect an enterprise's production environment. Most supply chain risk models reason only about direct dependencies — a malicious package imported by an application. Cascade depth analysis considers second-, third-, and higher-order effects: a compromised extension that harvests credentials for a package registry, which enables publication of a malicious package, which is imported by a security scanner, which runs with elevated permissions in a production CI/CD environment, which ultimately produces signed artifacts that enterprise systems treat as trusted.

Applying the Model to GlassWorm and TeamPCP

Analyzed through the Cascading Supply Chain Risk Model, both campaigns represent high-cascade-depth attacks:

Dimension	GlassWorm	TeamPCP / Mini Shai-Hulud
Primary node targeted	IDE extension ecosystem (OpenVSX/VS Code)	Security tooling, AI infrastructure, CI/CD pipelines
Node criticality	High — affects all developers in affected environments	Very high — affects artifact verification layer
Primary propagation vector	Credential harvesting → package publishing access	CI/CD runner compromise → signing key access
Secondary propagation	Published packages infect downstream installations	Signed malicious artifacts trusted by downstream consumers

Dimension	GlassWorm	TeamPCP / Mini Shai-Hulud
C2 resilience	Very high (blockchain-anchored)	High (automated, distributed campaign)
Cascade depth	3–4 levels (IDE → credentials → packages → consumers)	4–5 levels (tooling → secrets → pipeline → artifacts → production)
Detection challenge	Invisible Unicode obfuscation, extension trust model	Genuine SLSA attestations for malicious artifacts

The model highlights a property that the event-centric approach misses: by targeting nodes with high criticality and exploiting propagation vectors that traverse trust boundaries, these actors achieve cascade depth that is disproportionate to the initial attack complexity. The developer ecosystem's interconnection – which exists to facilitate collaboration and efficiency – becomes the mechanism of propagation.

Systemic Vulnerabilities in the Developer Ecosystem

The Extension Marketplace Trust Problem

IDE extension marketplaces represent a structural vulnerability that the industry has not adequately addressed. Unlike application stores for consumer software, extension marketplaces for developer tools historically applied minimal vetting. Extensions typically execute with the full permissions of the running IDE process, which in a developer context means access to local credential stores, configuration files, repository checkouts, and the terminal environment. The volume of available extensions – VS Code alone hosts tens of thousands – makes comprehensive review impractical, and the update cadence of actively maintained extensions creates a continuous publication stream that no static vetting process can fully monitor.

The extension marketplace trust problem is compounded by the developer habit of installing extensions on recommendation without reviewing the underlying code. Security reviews, code audits, and change monitoring – practices that developers routinely apply to code they ship – are rarely applied to the development tooling itself. This asymmetry reflects a blind spot: developers view their tooling as the instrument of security work rather than as an attack surface subject to the same scrutiny.

Open-Source Registry Integrity

The attacks on npm, PyPI, and Docker Hub documented in 2025–2026 exploit the fundamental design assumption of public package registries: that the entity publishing under a given namespace is the legitimate maintainer of that project. This assumption breaks down whenever a maintainer's publishing credentials are compromised. Because credential harvesting is precisely the primary payload of campaigns like GlassWorm, a feedback loop exists: IDE extensions harvest credentials, which enable unauthorized publication, which introduces malicious packages that harvest more credentials from developers who install them.

Most package registries have implemented some form of multi-factor authentication and have piloted attestation mechanisms that can verify build provenance. However, adoption of these protections is uneven, and as Mini Shai-Hulud demonstrated, even strong attestation mechanisms can be defeated when the build environment itself is compromised. The challenge is not purely technical; it also involves governance. Open-source registries are typically operated by non-profit foundations or corporate entities with limited resources for security infrastructure, and they face the difficult economics of providing free, ubiquitous service while bearing increasing security obligations.

CI/CD Pipelines as High-Value Targets

Continuous integration and continuous deployment pipelines are attractive targets because they aggregate secrets across organizational contexts. A single CI/CD runner may have access to source code repositories, artifact registries, cloud deployment credentials, code signing keys, database passwords, and API keys — often all simultaneously present as environment variables accessible to any code the pipeline executes. When security tooling installed in that pipeline is itself malicious, or when a malicious dependency is imported during the build process, the attacker gains access to all of these secrets at once.

CSA's DevSecOps research identifies CI/CD pipeline security as one of the highest-leverage controls in the software development lifecycle [20]. Yet in practice, pipelines often accumulate overprivileged credentials over time, as teams prioritize development velocity over least-privilege design. Secrets are long-lived, scoped more broadly than individual tasks require, and often replicated across multiple pipelines without a clear owner responsible for rotation. The same organizational pressures that lead to developer workstation credential sprawl — urgency, convenience, the assumption that internal tooling can be trusted — apply with compounded force to CI/CD environments.

The Verification Stack Paradox

Perhaps the most consequential systemic vulnerability exposed by the 2025–2026 campaigns is what might be called the verification stack paradox: the tools enterprises use to verify supply chain integrity — scanners, attestation verifiers, signing tooling, dependency analyzers — are themselves supply chain components, and their integrity is rarely verified with the same rigor they apply to other software.

If Trivy, a widely deployed container vulnerability scanner, is compromised and reports that a malicious container image is clean, the organization's supply chain verification process has been inverted: it now actively conceals threats rather than detecting them [10]. If a compromised signing tool produces attestations for malicious artifacts, the attestation mechanism — which exists to enable trust — becomes the mechanism of deception. The verification stack paradox means that investments in supply chain controls provide diminishing returns as attackers move upstream in that stack: the more enterprises rely on a specific verification tool, the more valuable that tool is as a target.

The AI Dimension

AI Tooling as a New Attack Surface

The compromise of LiteLLM in the TeamPCP campaign extends the supply chain attack surface into a relatively new domain: AI inference infrastructure [10]. LiteLLM, as an AI gateway and proxy, sits between enterprise applications and the large language model providers they consume. A compromised LiteLLM installation could intercept prompt content, inject modified responses, exfiltrate the sensitive data passed to AI models by enterprise workflows, or redirect inference traffic in ways that are difficult to detect without deep inspection of the AI layer.

This attack vector is architecturally distinct from traditional supply chain compromises because the payload may be purely semantic — a modification that does not introduce detectable malicious code but instead subtly alters AI model outputs in ways that manipulate downstream decision-making. As enterprises increasingly rely on AI-assisted workflows for tasks including code review, vulnerability triage, and security analysis, compromise of the AI inference infrastructure provides an attacker with leverage over the AI-augmented cognition of the organization itself.

AI-Accelerated Supply Chain Attacks

The inverse relationship also warrants analysis: AI tooling can accelerate the supply chain attack cycle for adversaries. The speed of the TanStack campaign — 84 malicious artifacts published across 42 packages in approximately six minutes — reflects automation that may incorporate AI assistance in identifying naming patterns, constructing valid package manifests, and coordinating multi-repository publication [16]. The ability to enumerate potential targets, construct plausible package names for typosquatting, and craft malicious packages that evade static analysis has been materially enhanced by generative AI capabilities accessible to threat actors.

MAESTRO, CSA's threat modeling framework for agentic AI systems, identifies supply chain security as a concern at Layer 1 (Foundation Models), where model training supply chains may be tampered with, and at Layer 5 (External Systems Integration), where AI agents interacting with external APIs and package ecosystems may trigger or be affected by supply chain compromises [21]. The confluence of AI tooling as a target and AI capabilities as a force multiplier for attackers creates a compounding risk dynamic that existing supply chain security frameworks, designed primarily for the pre-AI era, do not fully address.

Defensive Frameworks and Mitigations

Immediate Actions

Organizations should treat developer workstations, CI/CD environments, and security tooling as components of their critical asset inventory requiring continuous monitoring and integrity verification. The following actions address the most acute exposures revealed by the GlassWorm and TeamPCP campaigns.

Credential hygiene for developer environments warrants immediate attention. Organizations should audit and revoke stale GitHub, npm, and PyPI tokens, enforce short token lifetimes where the registry supports it, and require that publishing credentials be stored in hardware-backed credential managers rather than configuration files. The GlassWorm credential harvesting model depends on long-lived tokens stored in accessible locations; reducing credential lifetime and improving storage security directly limits the campaign's effectiveness.

Extension and plugin inventories should be treated as software assets subject to the same controls as production dependencies. Organizations should inventory installed IDE extensions across developer workstations, establish a registry of approved extensions, and implement policy enforcement that prevents installation of unapproved extensions in managed environments. New extension installations should trigger the same review workflow applied to new external dependencies.

Security tooling used in CI/CD pipelines — scanners, analyzers, signing tools — should be pinned to verified versions by digest rather than by mutable tags, and those digests should be updated through an explicit change management process rather than automatically on each pipeline run. The compromise of security tooling is particularly dangerous precisely because these tools often execute with elevated permissions and their outputs are trusted by downstream decisions; the same caution applied to production dependencies should apply to tooling with pipeline-level access.

Short-Term Mitigations

Within 30 to 90 days, organizations should pursue structural improvements to their CI/CD secret management practices. Secrets should be scoped to the minimum permissions required for each specific pipeline stage, should be rotated on a defined schedule, and should not be shared across unrelated pipelines. Organizations should evaluate whether their existing CI/CD architecture allows a compromised scanner to access cloud deployment credentials; in most default configurations, it does.

SBOM generation should be treated as a standard output of every build process rather than an optional or aspirational capability. Software Bill of Materials documents, paired with artifact attestations under frameworks such as SLSA or in-toto, create the traceability necessary to scope and respond to supply chain incidents quickly. When a compromised component is disclosed, organizations with comprehensive SBOMs can identify every affected artifact within their estate; organizations without them face a substantially more difficult scoping exercise. CSA's DevSecOps guidance and the joint CISA-NSA SBOM publication provide detailed implementation guidance [20][22][23].

The limitations of SLSA attestations exposed by Mini Shai-Hulud should prompt organizations to adopt defense-in-depth rather than treating any single attestation mechanism as a sufficient trust signal. Attestations verify process integrity; they do not verify the security of the build environment itself. Organizations should pair attestation verification with independent behavioral scanning of artifacts and should treat attestations as one input into a trust decision, not the sole input.

Strategic Considerations

At a strategic horizon of six months to two years, enterprises should reconsider how they reason about trust in developer tooling ecosystems. The conventional security posture treats the software development environment as a trusted domain — a location from which security controls are applied to other, less trusted domains. The campaigns analyzed in this paper demonstrate that this posture is untenable. Developer workstations, IDE extensions, CI/CD runners, and security tooling must be modeled as components of an adversarial environment and secured accordingly.

Applying Zero Trust principles to developer infrastructure means extending continuous verification and least-privilege access controls to the development environment itself, not only to production systems. This requires identity-based access to CI/CD infrastructure, ephemeral runner environments that do not persist secrets across pipeline runs, and behavioral monitoring of developer tooling that can detect anomalous credential access patterns consistent with harvesting activity [24].

Organizations that consume widely used open-source security tools should consider participating in the upstream security communities for those tools. Trivy, KICS, and similar projects operate as public goods; they benefit from enterprise investment in their security practices just as enterprises benefit from consuming them. This may include funding for maintainer security audits, contribution of security testing infrastructure, or participation in vulnerability disclosure programs. The security of the open-source security tooling ecosystem is a collective action problem that individual enterprise self-interest alone will not solve.

CSA Resource Alignment

MAESTRO Framework

CSA's MAESTRO (Multi-Agent Environment, Security, Threat, Risk, and Outcome) framework for agentic AI threat modeling provides directly applicable guidance for organizations deploying AI tooling in development pipelines. The compromise of LiteLLM documented in the TeamPCP campaign maps to MAESTRO's Layer 5 (External Systems Integration), which addresses risks associated with AI agents or gateways interacting with external services including package registries and build systems. MAESTRO's supply chain threat categories at Layer 1 (Foundation Models) are also relevant when AI-assisted code generation or review tools source their models through potentially compromised channels [21].

The MAESTRO framework's emphasis on blast radius analysis — assessing how far an agent's privileges extend beyond its required scope — applies directly to the CI/CD context: AI-integrated build tooling should be analyzed for the breadth of secrets and systems it can access, and that breadth should be minimized by architectural design.

AI Controls Matrix (AICM)

CSA's AI Controls Matrix provides a structured set of controls applicable to AI/ML supply chain security. The Model Security (MDS) domain includes controls addressing training pipeline security and model artifact scanning that are relevant for organizations evaluating the security of AI components in their supply chains,

including AI-assisted development tools. The Application and Interface Security (AIS) domain's controls for input validation and sandbox isolation apply to AI gateway components such as LiteLLM, which should be treated as a boundary requiring both inbound and outbound traffic inspection [25].

The AICM's coverage of shared responsibility across the cloud service provider, model provider, operator, and application developer roles is particularly useful for analyzing AI tooling supply chain risk: the compromise of an AI gateway affects the security responsibilities across all these roles simultaneously.

DevSecOps Framework

CSA's Six Pillars of DevSecOps guidance establishes an integrated framework for embedding security throughout the software development lifecycle. The supply chain security themes of the DevSecOps framework — software composition analysis, CI/CD pipeline security, vulnerability management for dependencies, and collective responsibility for third-party components — are directly applicable to the defensive mitigations recommended in this paper [20]. The Cascading Supply Chain Risk Model's emphasis on node criticality aligns with the DevSecOps framework's risk-tiering approach, which prioritizes security investment based on the blast radius of potential failures.

Zero Trust Architecture

CSA's Zero Trust guidance, including the Context-Based Access Control (CBAC) framework, provides architectural patterns for extending continuous verification into developer and build environments [24]. The principle of never-trust, always-verify applies with particular force to CI/CD environments, where the historical assumption of implicit internal network trust has been systematically exploited. Short-lived credentials, ephemeral execution environments, and workload identity verification — all elements of mature Zero Trust architecture — are the structural controls best suited to address the secrets-harvesting attack model demonstrated by both GlassWorm and TeamPCP.

STAR for AI

CSA's STAR (Security Trust Assurance and Risk) program, in its forthcoming AI-specific certification track, provides a mechanism for organizations to evaluate and communicate the security posture of AI-related software components, including development tooling. The Cascading Supply Chain Risk Model's treatment of verification tooling and AI gateways as high-criticality nodes in the supply chain suggests that STAR certification for these components would provide meaningful trust signals to enterprise consumers.

Conclusions

The developer ecosystem — the aggregate of registries, marketplaces, CI/CD pipelines, security tooling, and development environments through which all software passes — functions as critical infrastructure. Its interconnected trust relationships, which exist to make collaborative software development efficient and scalable, are simultaneously the propagation vectors through which supply chain attacks achieve their outsized impact.

GlassWorm and TeamPCP represent a maturation of the supply chain threat that demands a corresponding maturation in the defensive model. Self-propagating malware in IDE ecosystems, coordinated multi-registry attacks, defeat of SLSA provenance attestations, and compromise of security tooling itself are not isolated incidents but symptoms of a threat actor community that has systematically mapped the developer ecosystem's trust topology and is exploiting its structural properties.

The Cascading Supply Chain Risk Model proposed in this paper — framing supply chain risk as a function of node criticality, propagation vectors, and cascade depth — provides a conceptual basis for prioritizing defensive investment. Organizations that reason only about direct dependencies in their application code, while ignoring the security of their development tooling and CI/CD infrastructure, are addressing a fraction of their actual supply chain attack surface.

The practical implication is both organizational and architectural. Organizationally, developer workstations and build environments must be governed as assets within the security program, not as a trusted domain from which security work is performed on other assets. Architecturally, Zero Trust principles, ephemeral credentials, comprehensive SBOMs, and defense-in-depth in artifact verification must be applied to the development tier with the same rigor currently applied to production systems. The developer ecosystem is not where security begins; it is where security can be most consequentially undermined.

References

[1] CISA. "CISA's 2025 Year in Review: Driving Security and Resilience Across Critical Infrastructure." CISA, 2026.

[2] The Hacker News. "Developer Workstations Are Now Part of the Supply Chain." The Hacker News, May 2026.

[3] Sonatype. "2026 State of the Software Supply Chain." Sonatype, 2026.

[4] NSA. "AI and Machine Learning – Supply Chain Risks and Mitigations." NSA and Allied Nations Joint Publication, March 2026.

[5] CISA. "Software Bill of Materials (SBOM) for Cybersecurity." CISA, 2025.

[6] Koi Security. "GlassWorm: The First Self-Propagating Worm Using Invisible Code Hits OpenVSX Marketplace." Koi Security Blog, October 2025.

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

[8] BleepingComputer. "GlassWorm Malware Hits 400+ Code Repos on GitHub, npm, VS Code, OpenVSX." BleepingComputer, March 2026.

[9] Cloud Security Alliance. "CSA Research Note: GlassWorm Returns, Slices Back into Developer Toolchain." CSA AI Safety Initiative, 2026.

[10] Palo Alto Unit 42. "TeamPCP Supply Chain Attacks." Palo Alto Networks, 2026.

[11] Wiz. "Tracking TeamPCP: Investigating Post-Compromise Attacks Seen in the Wild." Wiz Blog, 2026.

[12] Okta Threat Intelligence. "Defending Against TeamPCP Software Supply Chain Attacks." Okta, 2026.

[13] Trend Micro. "Analyzing TeamPCP Supply Chain Attacks." Trend Micro Research, May 2026.

[14] GitGuardian. "Three Supply Chain Campaigns Hit npm, PyPI, and Docker Hub in 48 Hours." GitGuardian Blog, April 2026.

[15] SecurityWeek. "Over 320 npm Packages Hit by Fresh Mini Shai-Hulud Supply Chain Attack." SecurityWeek, 2026.

[16] Tenable. "Mini Shai-Hulud: Frequently Asked Questions." Tenable Blog, 2026.

[17] OpenSSF. "SLSA: Supply-chain Levels for Software Artifacts." OpenSSF, 2023.

[18] Chen, Z. et al. "Cascading Failure Modeling and Resilience Analysis of Supply Chain Networks." MDPI Systems, 2025.

[19] CISA, NSA. "Defending Against Software Supply Chain Attacks." CISA, April 2021.

[20] Cloud Security Alliance. "The Six Pillars of DevSecOps: Achieving Reflexive Security Through Integration." CSA, 2023.

[21] Cloud Security Alliance. "MAESTRO: Multi-Agent Environment, Security, Threat, Risk, and Outcome Framework." CSA AI Safety Initiative, 2025.

[22] CISA, NSA, and Partners. "Software Bill of Materials (SBOM) for Cybersecurity: Guidance for Enterprises." CISA, September 2025.

[23] GitHub Security. "Strengthening Supply Chain Security: Preparing for the Next Malware Campaign." GitHub Blog, 2026.

[24] Cloud Security Alliance. "Context-Based Access Control for Zero Trust." CSA Zero Trust Working Group, 2025.

[25] Cloud Security Alliance. "AI Controls Matrix v1.0." CSA, 2025.

