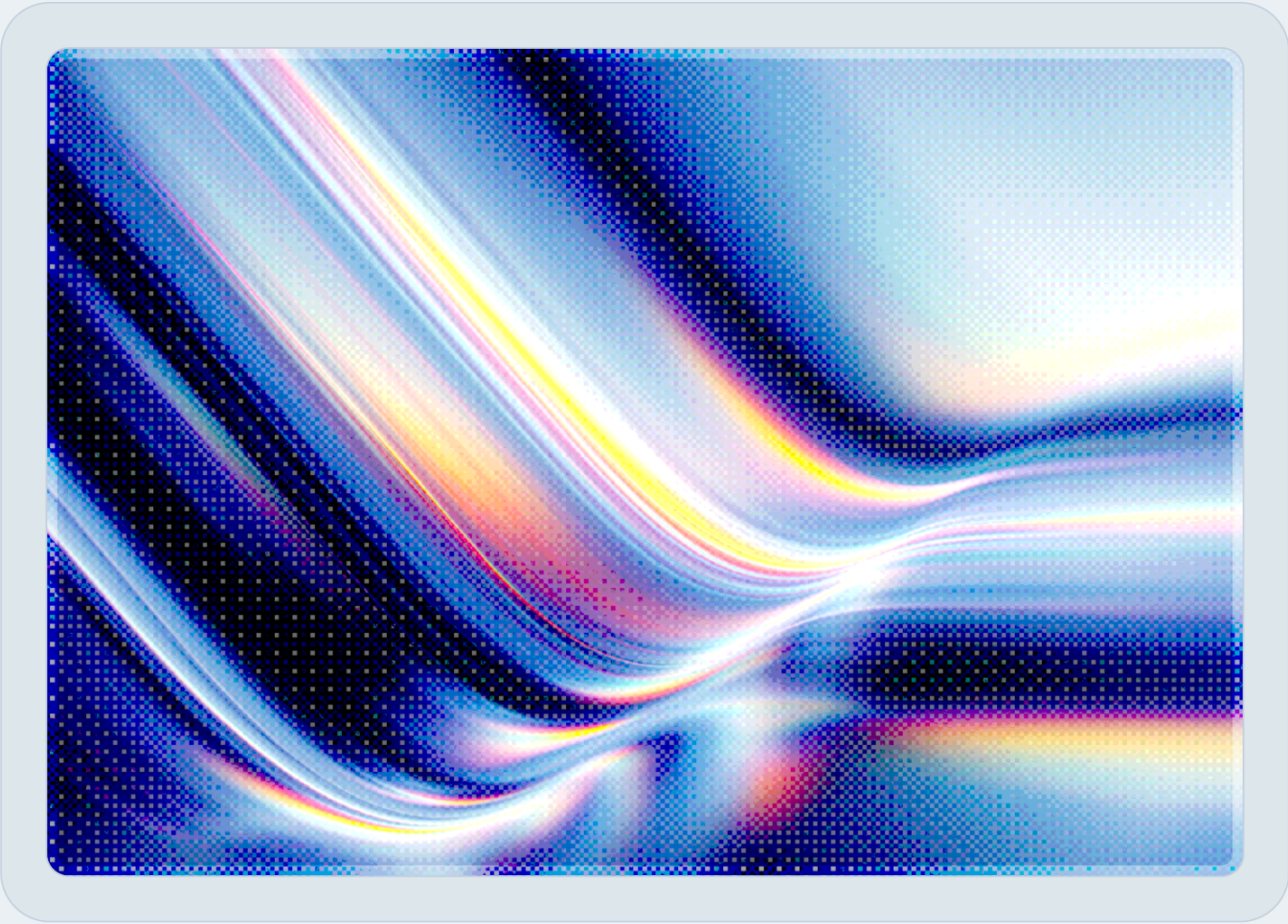
CSAI Foundation | Cloud Security Alliance

AsyncAPI npm Compromise: CI/CD Bypass Delivers Miasma RAT

2026-07-16

AI-assisted Rapid Research



CSAI

CSA cloud security alliance®

Key Takeaways

On July 14, 2026, unidentified threat actors published five trojanized versions of four packages in the widely used `@asyncapi` npm namespace, together accounting for an estimated 2 million weekly downloads [1][8]. The attackers did not forge or steal an npm publishing token directly; instead, they exploited a `pull_request_target` vulnerability in a separate docs-preview GitHub Actions workflow to obtain a privileged repository bot token, then used that stolen credential to push malicious commits directly onto pre-production branches of two AsyncAPI GitHub repositories, bypassing the branch-protection and peer-review controls applied to each project's main branch [2][4]. Because the commits landed through an authenticated automation identity rather than a maintainer account, each repository's own legitimate release workflow ran as designed, building and publishing the compromised packages with valid npm OIDC trusted-publisher provenance [1][3][4]. The injected code executed the moment a consuming application imported the library, rather than at install time, which means the common defensive advice to run `npm install --ignore-scripts` offered no protection [2]. The payload retrieved a multi-stage botnet framework known as Miasma from IPFS and other decentralized channels, establishing credential-harvesting and remote-access capability across compromised developer machines and build systems [1][3]. CSA assesses that this incident is best understood not as an isolated npm compromise but as the latest instance of a pattern -- following comparable attacks against Bitwarden's CLI, the TanStack ecosystem, an @antv package wave, and Red Hat's cloud services packages earlier in 2026 -- in which the trust automated release pipelines place in their own scoped credentials, rather than a maintainer's personal credentials, has become the primary target for open source supply chain intrusions [5].

Background

AsyncAPI is an open source specification and tooling project used to describe event-driven and message-based APIs, and its generator and specification packages are widely embedded as build-time dependencies across the Node.js ecosystem; researchers investigating the incident put combined weekly downloads across the four affected packages at roughly 2 million, with `@asyncapi/specs` accounting for the largest share of that total [1][8]. The compromise affected `@asyncapi/generator@3.3.1`, `@asyncapi/generator-helpers@1.1.1`, and `@asyncapi/generator-components@0.7.1`, all published from the

`asyncapi/generator` repository, along with `@asyncapi/specs@6.11.2` and a corresponding alpha release, published from the separate `asyncapi/spec-json-schemas` repository [3][4]. Because these are foundational tooling packages rather than end-user applications, the exposure could have extended into the build pipelines and development environments of any downstream project that resolved one of the poisoned versions. Kong's Insomnia, an open source API client that depends on `@asyncapi/specs`, illustrates the audit response this required: after checking its dependency tree against the malicious version list, the project published an advisory confirming that its own resolved version, `@asyncapi/specs@6.11.1`, was not among the compromised releases and that it was therefore not affected [6].

According to Microsoft's investigation, the entry point was a `pull_request_target` vulnerability in a separate docs-preview GitHub Actions workflow, a known anti-pattern in which a workflow checks out pull-request code while retaining access to repository secrets; an external researcher had flagged the underlying misconfiguration months earlier, and a proposed fix remained unmerged at the time of exploitation [2]. Exploiting that flaw allowed the attackers to steal a privileged repository bot token, rather than a maintainer's own npm or GitHub credentials, which they then used to push malicious commits directly onto branches that sat outside the branch-protection and peer-review controls applied to each project's main branch: `next` in the generator repository and a comparable pre-release branch in the specs repository [2][3][4]. Commits were attributed to a placeholder git identity, "Your Name you@example.com," a detail that suggests either a misconfigured automation credential or an attacker who did not bother to disguise the intrusion because the stolen bot token already granted sufficient reach [4]. Once the malicious commits landed, each repository's existing GitHub Actions release workflow ran as designed, built the packages, and published them to npm using the projects' registered OIDC trusted-publisher integration, so the resulting packages carried the same cryptographic provenance attestations as any legitimate release [1][3]. The common thread across the available accounts is that CI/CD automation and the scoped credentials it holds, rather than a maintainer's individual npm or GitHub account, functioned as the point of compromise, a pattern CSA has documented as "poisoned pipeline execution" in its prior supply chain security guidance [7]. Public reporting on this incident is limited to third-party security vendors; at the time of writing, neither affected repository had published its own GitHub Security Advisory or a maintainer post-mortem, so the account above should be read as third-party incident-response reconstruction rather than a confirmed first-party narrative.

Security Analysis

The obfuscated loader code was not delivered through npm lifecycle hooks such as `preinstall` or `postinstall`, which many organizations already treat as a red flag and block via install-time policy. Instead, the malicious logic was woven directly into legitimate source files that the packages already shipped and that consuming applications import in the ordinary course of use, including files such as `src/utils.js`, `lib/utils/ErrorHandling.js`, and `lib/templates/config/validator.js` [3]. Because the trigger condition was module load (a `require()` or `import` call) rather than package installation, the code executed inside the memory space of any process, including CI runners, developer laptops, and production build servers, that merely used the library for its intended purpose. Microsoft's threat intelligence team, which investigated the campaign, characterized this as import-time payload delivery and noted explicitly that it defeats `--ignore-scripts` and other install-hook-focused mitigations that many security teams rely on as a first line of defense against malicious packages [2].

Once triggered, the loader launched a detached, hidden Node.js child process that fetched an encrypted second-stage archive from the IPFS distributed storage network, decrypting and unpacking it into an operating-system-appropriate hidden directory disguised to resemble a legitimate application data folder, for example `~/.local/share/NodeJS/` on Linux, the analogous `Application Support` path on macOS, and a `%LOCALAPPDATA%` path on Windows [3][4]. The unpacked payload is the Miasma framework, a substantial bundled Node.js application that Socket's analysis measured at approximately 3.09 megabytes and, per a separate detailed technical writeup, comprising roughly 92,000 lines of code across more than 700 internal modules [1][3][8]. Researchers who obtained and decrypted a copy of the payload found it self-identified internally with a version string suggesting active development, and documented that it exposes command-and-control functions for file listing, retrieval, upload, and deletion; arbitrary shell command execution; and self-update, alongside persistence mechanisms such as systemd user services on Linux and comparable registry or launch-agent techniques on Windows and macOS [3][4].

CSA assesses that among Miasma's many design choices, its communications architecture carries the greatest operational significance for defenders. Rather than relying on a single command-and-control domain or IP address that defenders could sinkhole or block, the framework was built with as many as six independent channels: a conventional HTTP/REST endpoint anchored on a single server, IPFS itself, Nostr relays, BitTorrent's distributed hash table, a libp2p GossipSub mesh, and an Ethereum smart contract used as a decentralized fallback registry for locating live infrastructure [1][4][5]. This redundancy means that blocking any single channel, including the primary HTTP command-and-control server identified in multiple reports at the IP address 85.137.53[.]71, does not reliably deny the attacker

access to already-compromised hosts [3][5]. Documented credential-harvesting targets included browser-stored passwords, SSH private keys, npm and GitHub CLI authentication tokens, cloud provider credentials, Kubernetes service account tokens, and cryptocurrency wallets, and at least one report described a capability aimed at poisoning or manipulating sessions in AI coding assistants running on infected developer machines, a detail directly relevant to organizations that have integrated AI-assisted development tooling into engineer workstations [3][5]. Multiple sources also reported that credential-harvesting, cryptocurrency-wallet targeting, and self-propagation modules capable of publishing further trojanized packages to npm, PyPI, and Cargo were present in the shipped code but toggled inactive, alongside anti-analysis logic that detects sandboxed environments, Russian-language system locales, and the presence of endpoint tools from vendors including CrowdStrike, SentinelOne, and Microsoft Defender [1][2][5]. Because those modules exist in the codebase even when dormant, CSA assesses that any host confirmed to have loaded an affected package version should be treated as though full compromise occurred, since dormant capability can be reactivated through the framework's self-update function without a second exposure event [3].

According to StepSecurity's incident timeline, the first malicious commit to the generator repository's `next` branch landed at approximately 06:58 UTC on July 14, 2026, with the three trojanized generator-family packages published between 07:10 and 07:11 UTC and the two specs-family packages published between roughly 08:06 and 08:30 UTC following a second intrusion into the separate repository around 07:56 UTC [4]. The same source reports that all five malicious versions were unpublished from the npm registry by approximately 11:18 UTC, giving an exposure window of a little over four hours for the generator packages and roughly three hours for the specs packages [4]. Community infrastructure-security tooling, notably GitHub Actions runtime-monitoring products, reported detecting the anomalous outbound network activity associated with the campaign and adding the associated command-and-control endpoints to shared blocklists during the exposure window, which some vendors credit with limiting onward impact for customers already running that monitoring [4]. Researchers tracking npm supply chain activity through 2026 have linked the C2 infrastructure's hosting pattern, including its use of a specific Dutch autonomous system, to a threat cluster sometimes referred to as TeamPCP, implicated in the June 2026 compromise of Red Hat's cloud services packages and, before that, intrusions affecting the AntV and TanStack ecosystems [1][5]. Unit 42 itself hedges this attribution, however, noting that the source code for an earlier related toolkit, "Mini Shai-Hulud," leaked publicly in May 2026 and has enabled copycat actors to reuse the same techniques, so the shared infrastructure pattern is consistent with either a single actor iterating across multiple campaigns or a distinct group building on leaked tooling [5].

Recommendations

Immediate Actions

Organizations should first determine whether any system, including CI/CD runners, build agents, or developer workstations, resolved `@asyncapi/generator@3.3.1`, `@asyncapi/generator-helpers@1.1.1`, `@asyncapi/generator-components@0.7.1`, `@asyncapi/specs@6.11.2`, or `@asyncapi/specs@6.11.2-alpha.1` at any point between July 14 and July 15, 2026, by reviewing package-manager lockfiles, dependency-resolution logs, and internal or third-party registry mirrors and caches that may have retained the malicious versions after npm's own removal [3][4]. Any host where import (not merely installation) of one of these versions is confirmed should be treated as compromised: security teams should inspect for the indicators reported by multiple investigators, including outbound connections to IP address 85.137.53[.]71 on ports 8080, 8081, or 8091, requests to IPFS gateways, unexpected BitTorrent DHT or Nostr relay traffic, hidden files such as `sync.js` in application-data directories, and unauthorized systemd user services or scheduled tasks associated with persistence [3][4][5]. Where compromise is confirmed or cannot be ruled out, organizations should rotate every credential type the malware is documented to target, including npm tokens, GitHub personal access tokens and CLI sessions, SSH keys, cloud provider credentials, Kubernetes service account tokens, and browser-stored passwords, and should regenerate dependency lockfiles to force clean re-resolution rather than assuming a version bump alone remediates the exposure [4][5].

Short-Term Mitigations

Because this attack activated at import time and specifically evaded install-hook-based defenses, organizations should not rely solely on `--ignore-scripts` policies as a supply chain control and should instead prioritize pinning dependencies to known-good, cryptographically verified versions and validating package provenance attestations rather than trusting provenance as a substitute for content review [2]. Security and platform engineering teams should audit their own CI/CD configurations for the anti-patterns implicated in this incident, most notably any workflow that grants elevated permissions or secret access to code checked out from a pull request or a branch outside the primary protected branch, and should extend branch-protection and mandatory peer-review policies to every branch capable of triggering a publish or release workflow, not solely to the default branch [1][2]. Given that the malware's disabled modules include lateral propagation to PyPI and Cargo in addition to npm, organizations maintaining dependencies across multiple language ecosystems should apply equivalent pipeline-hardening review across all of them rather than treating this as an npm-specific problem [1][5].

Strategic Considerations

This incident reinforces a trend CSA has tracked across several 2026 supply chain intrusions in which attackers increasingly target the credentials held by automated release pipelines rather than a maintainer's own personal credentials, since a pipeline compromise inherits whatever legitimate signing, provenance, and distribution privileges the pipeline itself holds. Organizations that depend on open source tooling as part of their software development lifecycle should treat CI/CD systems as high-value, high-privilege infrastructure warranting the same access-control rigor, least-privilege token scoping, and monitoring applied to production systems, rather than as auxiliary automation. The presence of AI-assisted developer tooling as an explicit target in this campaign is also a signal worth strategic attention: as engineering organizations adopt AI coding assistants and agentic development tools, the credentials, session state, and configuration data those tools accumulate on developer machines become an additional asset class that supply chain attackers are beginning to target directly, and organizations should extend endpoint monitoring and credential-hygiene programs to cover these tools rather than treating them as outside the traditional security perimeter.

CSA Resource Alignment

This incident illustrates several of the risks CSA's *Software Transparency: Securing the Digital Supply Chain* addresses, including its explicit discussion of poisoned pipeline execution as a named CI/CD risk category and its emphasis on software composition analysis, artifact provenance validation, and trusted-repository controls as mitigations [7]. The AsyncAPI compromise demonstrates precisely the gap that guidance warns about: valid provenance attestations and a legitimate release pipeline provided no assurance of package integrity once the pipeline's input, rather than its output, was manipulated, underscoring the paper's recommendation that organizations validate the content and origin of a build's inputs and not rely on downstream signing alone. The malware's targeting of AI-assisted developer tooling and its credential-harvesting scope also connect this incident to the CSA AI Controls Matrix (AICM) v1.1, whose Threat and Vulnerability Management and Application and Interface Security domains call for continuous monitoring of software components and access pathways that touch AI development environments [9]. Organizations building AICM-aligned control programs should treat build-time and import-time dependency execution, not just installation-time scanning, as within scope for the software supply chain risks those domains are designed to cover.

References

[1] The Hacker News. "Compromised AsyncAPI npm Packages Deliver Multi-Stage Botnet Malware." The Hacker News, July 2026.

[2] Microsoft Security Blog. "Unpacking the AsyncAPI npm Supply Chain Compromise and Import-Time Payload Delivery." Microsoft, July 15, 2026.

[3] Socket. "Compromised npm Packages in the AsyncAPI Namespace Deliver Multi-Stage Botnet." Socket.dev, July 2026.

[4] StepSecurity. "Coordinated AsyncAPI Supply Chain Attack: Miasma RAT Delivered via Compromised CI/CD Pipelines in Two Repositories." StepSecurity, July 2026.

[5] Palo Alto Networks Unit 42. "The npm Threat Landscape: Attack Surface and Mitigations." Palo Alto Networks, updated July 15, 2026.

[6] Kong. "Security Advisory: Malicious AsyncAPI Package Versions Published (Miasma RAT)." Kong/Insomnia GitHub Issues, July 2026.

[7] Chris Hughes. "Software Transparency: Securing the Digital Supply Chain." Cloud Security Alliance, 2025.

[8] OX Security. "AsyncAPI npm Organization Compromised, 2M Weekly Downloads Affected." OX Security, July 2026.

[9] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

