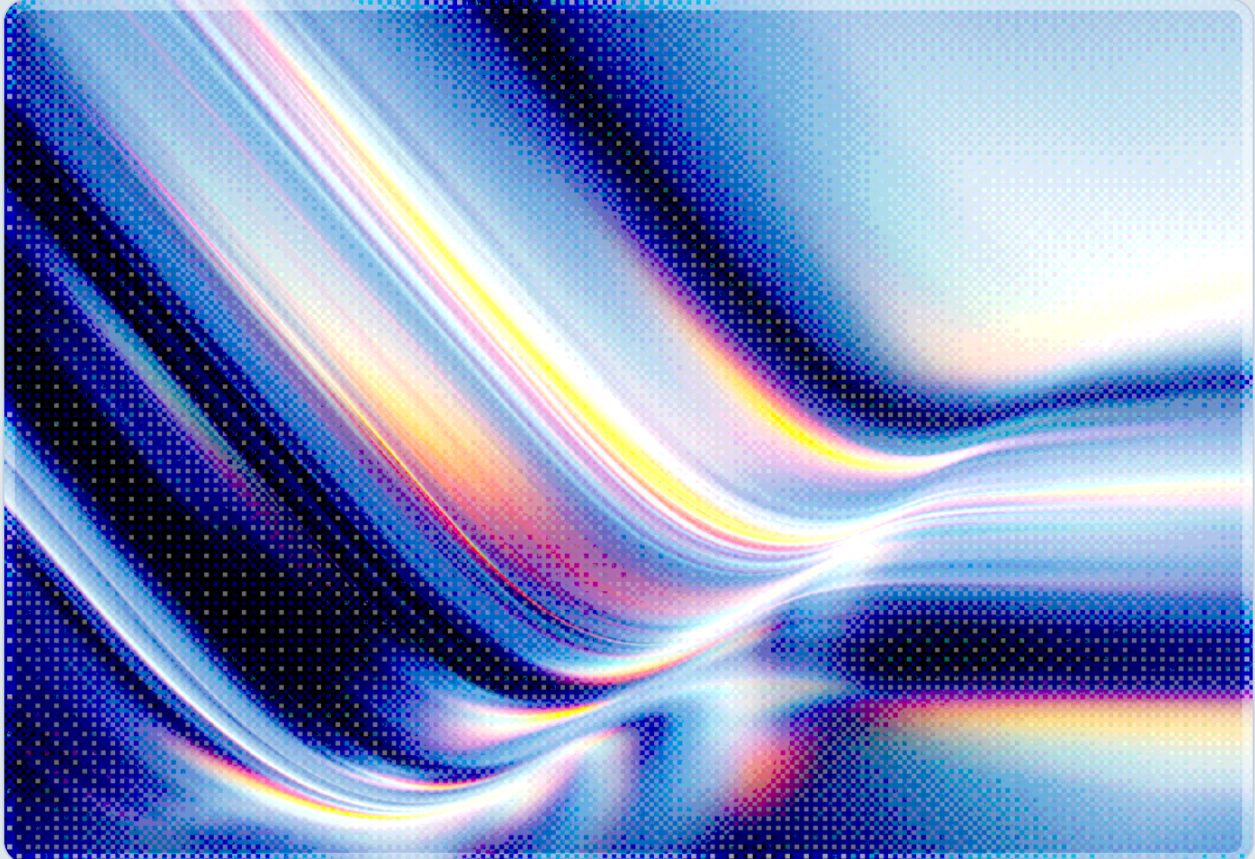


One Decoder, Many Copies: Brotli and the Vendored-Library Patch Tail

Vendored-library monoculture and the long patch tail in AI infrastructure after CVE-2026-111497

2026-10-26

 AI-assisted Rapid Research



CSAI

cloud
security
alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

CVE-2026-111497 is an out-of-bounds write in Google's Brotli C decoder, affecting upstream versions 1.0.3 through 1.2.0. It can be triggered by a crafted compressed stream, and an unauthenticated remote attacker who reaches the decoder may be able to crash the process, disclose memory, or execute code [1] [2]. The flaw has been called Node2Shell, HTTP2Shell, and Broccoli2Shell. Those names describe where it was first noticed, not where it lives. The vulnerable code is copied into language runtimes, proxies, databases, archivers, browsers, and firmware tooling, and each copy is patched on its own owner's schedule [3][5].

For organizations running AI workloads, the most important point is that vendored copies form the long tail of this vulnerability. Upstream Brotli, Node.js, Bun, Python's Brotli and brotlicffi packages, Envoy, and Istio shipped fixed releases between October 6 and October 23, 2026. As of October 25, however, a public tracker still listed the .NET runtime, DuckDB, ClickHouse, Apache Arrow's bundled build, and several archivers and firmware tools as unpatched [5]. The outbound path also reaches AI agents. HTTP clients that decode Brotli automatically, including Node.js `fetch()` and Python's `urllib3`, `requests`, `httpx`, and `aiohttp` when a Brotli package is installed, can be attacked by any server they contact, which puts agents, crawlers, and URL-fetching tools in scope [3][6].

Detection and disclosure compound the problem. Scanners can be wrong in both directions: patched Node.js releases still report the old Brotli version string, distributions backport fixes without changing upstream version numbers, and statically linked copies often do not appear under any Brotli package name at all [3][5]. The fix itself was public for about eight months before it was publicly identified as a security fix; it was committed to the upstream repository on February 27, 2026, and labeled as a security issue only with the October releases [3]. Finally, exploitation has been demonstrated but not yet observed. Researchers have shown code execution against unpatched Node.js, but as of October 24 no in-the-wild exploitation had been reported and the CVE was not in CISA's Known Exploited Vulnerabilities (KEV) catalog [3][14][15].

Background

Brotli is a general-purpose compression format that Google created and maintains as a C library. It is the standard `br` content encoding for HTTP, and it is also used inside file formats such as WOFF2 web fonts, JPEG XL images, and Parquet columnar data files. Brotli is small, permissively licensed, and has no external dependencies. Those qualities make it easy to copy straight into another project's source tree instead of linking a shared system library. Over a decade, that convenience has produced a monoculture. A single decoder implementation runs in dozens of independently maintained copies, and no single party can see or update all of them [5].

The vulnerability itself is a state-handling error. According to the Node.js advisory, the decoder reported an invalid block switch as a temporary need for more input instead of a permanent format error. A streaming caller therefore kept appending bytes to an eight-byte internal buffer and wrote past its end [1]. The GitHub advisory classifies the root cause as CWE-252 (Unchecked Return Value), reflecting the misreported decoder status, and the National Vulnerability Database also lists CWE-787 (Out-of-bounds Write) for the consequence; GitHub scores the issue 9.2 under CVSS v4.0 [2][13]. Node.js itself rates the issue High rather than Critical [1], so readers should expect severity ratings to differ by vendor and deployment context.

The disclosure timeline explains much of the current exposure. Tanium's reconstruction shows that the fix was committed to the google/brotli repository on February 27, 2026, without an advisory or release [3]. Envoy shipped fixed releases on October 6, and Istio's bulletin named the CVE on October 15. Node.js released versions 22.23.4, 24.22.1, and 26.12.1 on October 21, the same day upstream Brotli tagged versions 1.0.10, 1.1.1, and 1.2.1 [1][3][12]. The CVE record and the patched Python packages followed on October 22, and Bun and GitHub published advisories on October 23 [2][3]. Node.js credits the report to OpenAI Codex Security researchers Khai Tran and Sylvie Mayer [1]. Imperva's red team then demonstrated code execution on an unmodified official Node.js 24.22.0 Linux arm64 image, an exploit that took several attempts [3][14].

Security Analysis

Why one bug becomes many patch cycles

When a library is linked dynamically from the operating system, one package update fixes every consumer the next time it restarts. Debian's handling of this CVE shows the model: its Node.js and python-brotlicffi packages are built against the system Brotli library, so a fix to the `brotli` source package flows to

every consumer (although Debian still tracked its Node.js packages as needing a follow-on change) [7]. Vendored copies break that model. Each project that compiles Brotli into its own binary, wheel, or container image needs its own release, and its users must then install that release. A bundled-copy tracker published by Phoenix Security notes that operating system updates do nothing for these embedded decoders [5].

The table below summarizes publicly reported status as of October 24–25, 2026. It covers representative projects and is not a complete list. The tracker's authors note that GitHub code search skips submodules and some large repositories, and that private or build-time vendored copies are invisible to it [5].

Component	Role in AI infrastructure (illustrative)	Reported status	Source
Node.js 22.x / 24.x / 26.x	MCP servers, agent tooling, web front ends	Fixed in 22.23.4, 24.22.1, 26.12.1; end-of-life lines receive no fix	[1][3]
Python Brotli / brotlicffi	Transitive dependency of urllib3, requests, httpx, aiohttp	Fixed in 1.2.1 / 1.2.1.0	[2][6]
Bun	JavaScript runtime and single-file executables	Fixed in 1.4.4	[4]
Envoy / Istio	Model-serving ingress, service mesh, AI gateways	Fixed in listed releases; affected only where the Brotli decompressor is enabled	[3][4]
Go andybalholm/brotli	Go services and proxies	Fixed in v1.2.7	[4]
.NET runtime	HttpClient auto-decompression, ASP.NET Core request decompression	Unpatched as of Oct 25; bundles 1.1.0	[5]
DuckDB, ClickHouse, Apache Arrow (bundled	Analytics and Parquet ingestion for training,	Unpatched as of Oct 25; fixes tracked in upstream	[5]

Component	Role in AI infrastructure (illustrative)	Reported status	Source
build)	evaluation, and RAG data	PRs/issues	
7-Zip-zstd, p7zip, EDK II, woff2 (Makefile build)	Archive handling, firmware, font processing	Unpatched as of Oct 25	[5]

The table does not support a simple resourcing explanation for who patched first. Several of the early fixes came from projects that shipped ahead of the public CVE record, notably Envoy and Istio, while the community-maintained Go port also appears among the fixed entries [3][4]. The unpatched entries, by contrast, include both large vendor projects such as the .NET runtime and ClickHouse and smaller data-processing and file-handling tools. That mix suggests that notification timing and release cadence, not only resourcing, determine where the tail sits, and that a large share of it may lie in the components AI data pipelines use to ingest third-party content.

Exposure paths specific to AI workloads

Three distinct paths carry attacker-controlled bytes to the decoder in AI environments, and each calls for a different response. The inbound path is familiar. Any server that decompresses a **Content-Encoding: br** request body before authentication exposes the decoder to anonymous input. Envoy and Istio deployments are affected only where Brotli decompression is configured [3]. Teams that front model-serving endpoints with Envoy-based gateways, including the Agent Router project (formerly Envoy AI Gateway) [8], should check their filter configuration rather than assume exposure either way.

For agentic systems, the outbound path deserves particular attention, because it requires no inbound exposure at all. HTTP clients that advertise Brotli support decode responses automatically. In Python, this applies to urllib3 and therefore requests whenever Brotli or brotlicffi is installed, and those packages often arrive transitively without the application ever importing them [6]. Aikido reports roughly 82.8 million monthly downloads for the Python Brotli package alone [6]. An AI agent that browses the web, follows links in retrieved documents, or calls user-supplied webhook and tool URLs hands the decoder to whoever controls the destination server. Prompt injection is one way an attacker can steer an agent toward such a URL. This attack inverts the usual threat model: no inbound port needs to be open, and the attacker only has to be fetched. Tanium names scrapers, integration services, and developer tools as in scope, and the same outbound-fetch logic applies to AI agents [3][6].

The file path is the least visible of the three. The Phoenix Security tracker notes that the flaw is reachable through Brotli-compressed Parquet, JPEG XL, WOFF2, PDF, archive, and firmware inputs, not only HTTP bodies [5]. AI data pipelines routinely load Parquet datasets from public hubs, partners, and scraped sources into DuckDB, Arrow, or ClickHouse. Where those engines carry an unpatched vendored decoder, a poisoned dataset file could reach it. This path is inferred from the reported affected inputs; no public demonstration against a data pipeline has been reported.

Why detection is unreliable

Version-based detection fails in both directions for this CVE. Node.js continues to report `process.versions.brotli` as 1.1.0 or 1.2.0 on fixed releases, because the fix was cherry-picked without a version bump [3]. Linux distributions keep the upstream version number when they backport, so version-only scanners flag patched systems as vulnerable [5]. False negatives carry greater risk, because they produce false assurance. Statically linked copies do not appear under the `libbrotli` or `brotli` package names, and Phoenix Security reports that container and software composition analysis scanners typically miss them unless an advisory maps the CVE to the parent product [5]. Checking `/proc/*/maps` only reveals dynamically loaded libraries. The npm package named `brotli` is an unrelated library, so name matching can produce misleading results [3].

Two more reliable signals have emerged. In source trees, including submodules, a patched `decode.h` defines the error code `BROTLI_DECODER_ERROR_FORMAT_BLOCK_SWITCH`, so its absence marks an unpatched copy [3][5]. For prebuilt binaries and wheels, the only practical options are a software bill of materials (SBOM) that records the embedded component or an explicit statement from the vendor [5]. Wiz reports that 91% of cloud environments it observes contain resources vulnerable to the CVE [4]. That figure comes from the vendor's own telemetry and depends on its detection logic, but it indicates how widely the decoder is deployed.

The eight-month gap

The upstream fix sat in public source for about eight months without a release or advisory [3]. A silent fix in a widely vendored library creates an asymmetry. Attackers who diff upstream commits can find the bug, while downstream maintainers, who track releases and advisories rather than commits, have no prompt to pull the change. The report came from researchers on OpenAI's Codex Security team [1], which develops AI-assisted vulnerability discovery tools. CSA's earlier analysis of AI-driven discovery showed that disclosure volume is already outpacing maintainers' capacity to remediate [9]. CSA assesses that machine-speed discovery feeding into release-gated, vendor-by-vendor patching is likely to make long tails like this one more common.

Recommendations

Immediate Actions

Organizations should first update the components with released fixes: Node.js to 22.23.4, 24.22.1, or 26.12.1; Bun to 1.4.4; Python Brotli to 1.2.1 and brotlicffi to 1.2.1.0; Go's andybalholm/brotli to v1.2.7; and Envoy and Istio to their patched releases [1][2][4]. Rebuilding container images, regenerating lockfiles, recompiling single-executable applications, and restarting long-running processes are all necessary, because an updated package does nothing for a process that still has the old code in memory. Teams should verify the fix where the code runs, not in a package list. Node.js is a case in point: its reported Brotli version cannot confirm a fix there, so the Node.js release number is the reliable check [3].

Priority should go to AI agents and services that fetch arbitrary URLs, since they can be attacked without any inbound exposure. Where an agent runtime cannot be patched immediately, removing the Python Brotli and brotlicffi packages from that environment removes the decoder used by urllib3, requests, httpx, and aiohttp, at the cost of possible errors from servers that send Brotli anyway [6]. Other packages in the same environment may still carry their own vendored copies. Rejecting **Content-Encoding: br** request bodies at the edge reduces inbound exposure but does nothing for outbound fetches or file inputs [5][6].

Short-Term Mitigations

Over the coming weeks, security teams should search build trees, including submodules and third-party directories, for vendored **decode.h** files that lack **BROTLI_DECODER_ERROR_FORMAT_BLOCK_SWITCH** [5]. For prebuilt products, particularly .NET applications, analytics engines, and archive tools on the unpatched list, they should request SBOM data or written status statements from vendors. Untrusted Parquet and archive files should be processed in isolated, low-privilege workers until the engines that read them are confirmed patched. Agent egress should run through a controlled proxy that can strip or reject Brotli encoding while the client fleet is remediated. Scanner findings should be cross-checked against distribution package release numbers to avoid both false alarms and false assurance [3][5].

Strategic Considerations

The longer-term lesson is about inventory granularity. An SBOM that lists only top-level packages would have missed the statically linked and vendored copies that make up much of the unpatched exposure described here [5]. Procurement and internal build standards should require component-level SBOMs that record vendored and statically linked libraries, and suppliers should commit to response times for embedded-component CVEs, not only first-party ones. Where practical, teams should prefer linking a maintained system library over vendoring, so that one update can fix every consumer.

AI platform teams should also treat outbound content handling as attack surface. Agent security designs that focus only on prompt-level threats can overlook memory-safety risk in parsers and decompressors that run on fetched content before any model sees it. Finally, organizations that depend on small, widely vendored C libraries should consider supporting upstream maintainers and watching upstream commits for silent security fixes, since release-gated monitoring left an eight-month blind spot in this case [3][9].

CSA Resource Alignment

CSA's research note [Project Glasswing: AI Discovery Outpaces Open Source Patching Capacity](#) documented how AI-driven vulnerability discovery produced far more disclosures than maintainers could remediate, with only 97 of 1,596 disclosed findings fixed at the time of analysis [9]. CVE-2026-111497 shows the next stage of that problem. Even once an upstream maintainer fixes a reported bug, the fix has to cross every vendored copy before users are protected. Glasswing's recommendations on direct maintainer channels and on planning for the disclosure-to-patch window apply directly to the unpatched downstream projects identified here.

CSA's presentation [Software Transparency: Securing the Digital Supply Chain](#) addresses SBOM, VEX, SLSA, and supply chain transparency requirements [10]. The Brotli case is a concrete test of those requirements. Package-level inventories missed statically linked copies, and the most reliable remaining signals for prebuilt binaries were component-level SBOMs and vendor attestations [5]. Organizations can use this incident to check whether their SBOM requirements actually capture embedded third-party code.

The [AI Controls Matrix \(AICM\) v1.1](#) gives the control basis for operationalizing these recommendations [11]. Its Threat and Vulnerability Management domain covers vulnerability identification and remediation prioritization, which here includes inventorying embedded components and verifying fixes at runtime. Its Application and Interface Security domain covers secure handling of untrusted input at interfaces,

including the outbound fetch paths of AI agents. Its supply chain controls address supplier transparency obligations, which organizations can extend to component-level SBOMs and response commitments for embedded-component CVEs.

References

- [1] Node.js Project. "[Wednesday, October 21, 2026 Security Releases](#)." Node.js Blog, October 21, 2026.
- [2] GitHub. "[GHSA-9v8h-gggm-4m35: Out-of-bounds write in the Brotli decoder \("Node2Shell"\) also affects the PyPI Brotli package](#)." GitHub Advisory Database, October 22, 2026.
- [3] Tanium. "[HTTP2Shell: Find and Fix Vulnerable Brotli Copies with Tanium](#)." Tanium Blog, October 24, 2026.
- [4] Merav Bar, Wiz. "[Broccoli2shell: Everything You Need To Know About The Critical Brotli Decoder Vulnerability](#)." Wiz Blog, October 25, 2026.
- [5] Phoenix Security. "[HTTP2Shell Expansion \(CVE-2026-111497\): Where the Brotli Decoder Bug Went After Node.js and Python](#)." Phoenix Security Blog, October 25, 2026.
- [6] Dania Durnas, Aikido Security. "[Node2Shell Brotli vuln \(CVE-2026-111497\) evolves to HTTP2Shell, as libraries and apps using HTTP are affected](#)." Aikido Blog, October 23, 2026 (updated October 25, 2026).
- [7] Debian Security Team. "[CVE-2026-111497](#)." Debian Security Bug Tracker, accessed October 26, 2026.
- [8] Agentic AI Foundation. "[Agent Router \(formerly Envoy AI Gateway\)](#)." Agent Router, accessed October 26, 2026.
- [9] Cloud Security Alliance. "[Project Glasswing: AI Discovery Outpaces Open Source Patching Capacity](#)." CSA, June 7, 2026.
- [10] Cloud Security Alliance. "[Software Transparency: Securing the Digital Supply Chain](#)." CSA Event Presentation, October 13, 2022.
- [11] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1](#)." CSA, June 22, 2026.
- [12] Google. "[google/brotli Releases](#)." GitHub, accessed October 26, 2026.
- [13] NIST. "[CVE-2026-111497 Detail](#)." National Vulnerability Database, October 22, 2026.
- [14] Imperva. "[Node2Shell: From Brotli Data to Remote Code Execution](#)." Imperva Blog, October 2026.
- [15] CISA. "[Known Exploited Vulnerabilities Catalog](#)." Cybersecurity and Infrastructure Security Agency, accessed October 26, 2026.

