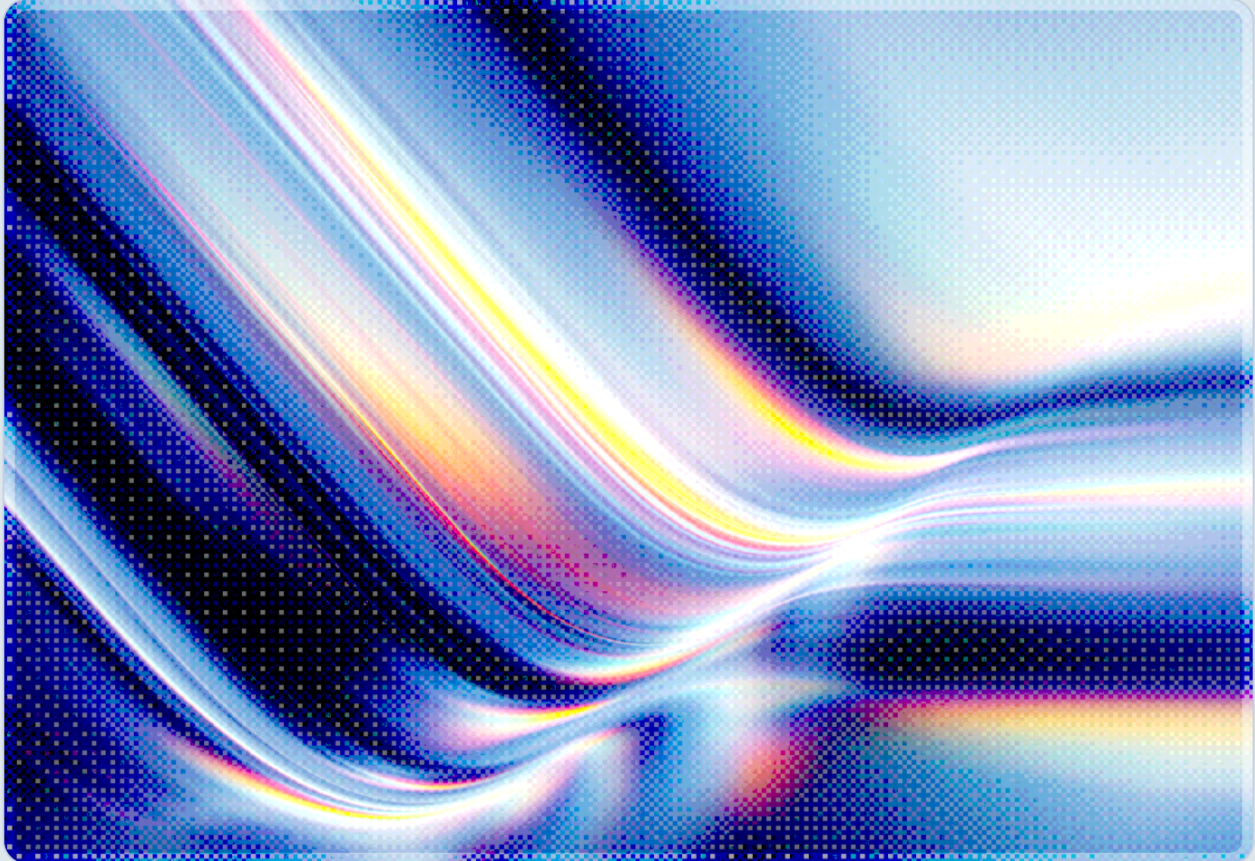


Brotli2Shell: A Compression Library Flaw Becomes Cross-Runtime RCE

Security Implications and Guidance for CVE-2026-111497
(Node2Shell / HTTP2Shell)

2026-10-28

 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 the decoder of Google's open-source Brotli compression library, affecting Brotli versions 1.0.3 through 1.2.0 [1][3]. Because Brotli is embedded in language runtimes, HTTP clients, proxies, and media libraries, one decoder bug reaches software built on very different stacks. Node.js, Bun, .NET, Envoy Proxy, Istio, Python HTTP clients, and PHP for Windows have all shipped fixes [2][5][8]. The flaw has been nicknamed Node2Shell, HTTP2Shell, Brotli2Shell, and Broccoli2Shell. Google, as the CVE Numbering Authority, scores it 9.2 under CVSS v4.0. Node.js rates it High, and Microsoft scores the .NET exposure at 8.1 under CVSS v3.1 [1][2][8]. Those ratings contrast with an assessment from Konfirmity, reported by Sysdig, that the upstream fix commit described the bug as low severity [11].

CISA added the vulnerability to its Known Exploited Vulnerabilities (KEV) catalog on October 27, 2026, with a November 10 remediation deadline for U.S. federal civilian agencies [4]. Public proof-of-concept exploits are available on GitHub [5][11]. Researchers have shown code execution against official, unmodified Node.js 24.22.0 builds on both arm64 and x86-64 [5][7]. Exploitation still depends on the target build and is not reliable, and the first in-the-wild attempt that ELLIO analyzed appeared to use a defective payload [7]. Crashes are much easier to trigger than shells, so denial of service is the most likely outcome of an attack today. Code execution has nonetheless been demonstrated on stock builds, and exploits may become more reliable as they mature, so organizations should not plan as though code execution is ruled out [5][7][11].

The upstream fix was committed to the Brotli repository on February 27, 2026, and sat publicly readable for roughly eight months before coordinated disclosure on October 21 [1][5][12]. Patching is harder than usual because many products bundle or statically link their own copy of Brotli. Updating the operating system's library does not fix Node.js, Bun, self-contained .NET applications, or container images. Each of these needs a fixed vendor release, a rebuild, and a restart [5][8]. The vulnerability was credited to OpenAI Codex Security [1][2], one of a growing number of AI-assisted findings in widely shared open-source code [15][18].

Organizations should immediately inventory every runtime and binary that carries a Brotli decoder and upgrade to fixed releases. They should then rebuild and redeploy containers and self-contained artifacts, and stop decompressing untrusted Brotli data anywhere a patch cannot be applied right away.

Background

Brotli is a general-purpose lossless compression format developed by Google and standardized for HTTP content encoding. Browsers, CDNs, and servers negotiate it through the `br` value in the `Accept-Encoding` and `Content-Encoding` headers. Its compression ratio has led to wide adoption for web traffic, and the reference C decoder has been copied, vendored, wrapped, and linked into many kinds of software: language runtimes, HTTP client libraries, service-mesh proxies, database engines, and media codecs. In most of these, Brotli is invisible to application developers. A framework decompresses a request body or an HTTP client decodes a response automatically, and the application only ever sees plaintext [5][6].

The flaw sits in block-switch handling inside the decoder (`c/dec/decode.c`). According to the Node.js advisory, an invalid block switch was reported as a temporary need for more input instead of a permanent format error. The streaming decoder therefore kept appending input bytes to an eight-byte internal buffer, writing past its end and corrupting adjacent decoder state [2]. The CVE record classifies the issue as both an out-of-bounds write (CWE-787) and an unchecked return value (CWE-252). It states that a remote, unauthenticated attacker can corrupt memory, disclose memory contents, cause denial of service, or potentially execute code with a crafted Brotli stream [1]. The upstream fix, commit `0ed4268` , makes the decoder fail immediately on a block switch when the stream declares only one block type [6][11].

The disclosure timeline is unusual and matters for risk assessment. Tenable's reconstruction, summarized in the table below, shows the fix landing upstream in February, downstream proxies shipping it quietly in early October, and the coordinated public disclosure arriving only on October 21 [5]. CISA's listing followed six days later [4].

Date (2026)	Event
February 27	Fix committed to the google/brotli repository with no release or advisory [1][5]
Early October	Envoy Proxy and Istio ship fixed releases without naming the flaw [5]
October 21	Node.js publishes its security advisory; Google releases Brotli 1.2.1, 1.1.1, and 1.0.10 [2][3]
October 22	CVE record published; patched Brotli and brotlicffi reach PyPI [1][10]

Date (2026)	Event
October 25	Microsoft publishes the .NET advisory and updates [8]
October 26	PHP ships updated Windows builds; ELLIO reports exploitation traffic [5][7]
October 27	CISA adds CVE-2026-111497 to the KEV catalog [4]

Security Analysis

Why One Decoder Bug Crosses Runtimes

The defining feature of this vulnerability is how far it spreads, not the bug class itself. Out-of-bounds writes in C parsers are common. What sets this one apart is that the same decoder code ships inside ecosystems that security teams usually inventory, patch, and monitor separately. In Node.js, the vulnerable code is bundled into the runtime itself and compiled from the `deps/brotli` directory of the Node.js source tree [19]. Updating application dependencies therefore does nothing, and only a fixed Node.js build removes the flaw [2][11]. Akamai, as cited by Tenable, has reported that its telemetry shows 61% of networks running affected Node.js versions, although neither source defines that population, and version presence alone does not indicate that untrusted Brotli input reaches the decoder [5]. In .NET, the decoder sits behind `BrotliStream`, `BrotliDecoder`, `HttpClient` automatic decompression, and ASP.NET Core request decompression. Microsoft notes that updating the machine-wide shared runtime does not replace a decoder already linked into a NativeAOT executable [8]. In Python, `urllib3`, `requests`, `httpx`, and `aiohttp` use the `brotli` or `brotlicffi` packages automatically whenever those packages are installed. Applications can be exposed through a transitive dependency without ever importing Brotli themselves [9][10].

JFrog's research shows that the exposure reaches beyond HTTP. A crafted JPEG XL file containing a malicious `brob` box reached the decoder through `libjxl` in `FFmpeg`, and JFrog's crash showed attacker control over the corrupted pointer. JFrog stopped short of claiming code execution [6]. In ClickHouse, a single HTTP request with `Content-Encoding: br` crashed the tested server. In Apache Arrow, a 534-byte input sent the decoder into an indefinite loop through Parquet or buffer decompression, with no network access required [6]. At the time of JFrog's publication, ClickHouse had merged a fix without releasing it, and the Arrow issue was still open [6]. The table below summarizes how the decoder is reached in each major ecosystem and which releases carry the fix.

Ecosystem	How the decoder is reached	Fixed versions	Patch notes
Google Brotli (C)	Direct library use	1.0.10, 1.1.1, 1.2.1 [3]	Programs dynamically linking the OS library pick up the fix on restart [5]
Node.js	<code>node:zlib</code> , <code>fetch()</code> response decoding, body-parsing middleware [2][6]	22.23.4, 24.22.1, 26.12.1 [2]	Lines 20, 23, and 25 are end-of-life and receive no fix [5]
Bun	Bundled decoder	1.4.4 [5]	Bundled copy; requires runtime upgrade
.NET 8/9/10/11	<code>BrotliStream</code> , <code>HttpClient</code> , ASP.NET Core request decompression [8]	8.0.33, 9.0.22, 10.0.14, 11 RC3 [8]	Self-contained and NativeAOT apps must be rebuilt; .NET Framework not affected [8]
Python	<code>urllib3</code> , <code>requests</code> , <code>httpx</code> , <code>aiohttp</code> via <code>brotli</code> / <code>brotlicffi</code> [9]	<code>brotli</code> 1.2.1, <code>brotlicffi</code> 1.2.1.0 [10]	Exposure can arrive through transitive dependencies [9]
Envoy / Istio	Brotli decompressor filter [5]	Envoy 1.36.12, 1.37.8, 1.38.6, 1.39.3, 1.40.0; Istio 1.29.9, 1.30.6, 1.31.2 [5]	Fixes shipped before public disclosure [5]
PHP for Windows	Bundled Brotli in Windows builds [5]	8.2.35-1, 8.3.36-1, 8.4.27-1, 8.5.12-1 [5]	PHP brotli extension fixed in 0.21.1 [5]

Attack Surface: Inbound and Outbound

Being exposed requires two things: a vulnerable decoder must be present, and untrusted input must reach it [6]. Researchers stress that a vulnerable version number alone does not establish exploitability [6][12]. The input can arrive in two directions, though, and the second is easy to overlook. On the inbound side, an attacker sends a request body marked `Content-Encoding: br` to a server or middleware that decompresses it. ELLIO obtained shells against a plain `node:http` server calling `node:zlib`, Express 5 with `express.json()`, and Koa 3 with `@koa/bodyparser` [7]. On the outbound side, an application fetches content from a server the attacker controls, and its HTTP client decodes a malicious Brotli response automatically. Webhook senders, link-preview generators, crawlers, AI agents with web-fetch tools, and any "retrieve this URL" feature fall into this category [6][7][9].

The outbound direction matters for mitigation planning. Blocking inbound `Content-Encoding: br` at a proxy or load balancer protects only the inbound path. Tenable notes that removing `br` from `Accept-Encoding` is not enough on the client side either, and that Brotli decoding must be disabled in the client itself [5]. This is a particular concern for agentic AI systems and retrieval pipelines built on Node.js or Python. These systems routinely fetch arbitrary, attacker-influenced URLs as part of normal operation, which places them on the outbound attack path by design.

Exploitation Status and Reliability

The evidence on exploitation is mixed, and organizations should read it carefully instead of relying on headline severity alone. Imperva demonstrated code execution against the official Node.js 24.22.0 Linux arm64 image after multiple attempts, and has not released its exploit [5][12]. ELLIO then reproduced code execution against unmodified x86-64 builds of Node.js 24.22.0 with address space layout randomization enabled, getting a shell at around the fifth attempt in one configuration [7]. ELLIO also documented the limits. Exploitation depends on a per-build memory offset: the 24.22.0 offset produced no results against 22.23.3 or 26.12.0 in 10,000 attempts, and most of roughly 400 attempts against Express crashed the worker instead of returning a shell [7].

On October 26, ELLIO's sensors recorded exploit traffic carrying a 23 MB Brotli-encoded request built for Node.js 24.22.0 on x86-64. It came from a source ELLIO associates with prior mass-exploitation and scanning activity [7]. In ELLIO's own tests, the attacker's chain did not get a shell in 50 attempts, and ELLIO characterized it as a likely first-generation attempt with a defective final stage [7]. CISA's ADP enrichment of the CVE record, dated October 23, rates exploitation as "active" and automatability as "no," and CISA added the CVE to KEV on October 27 [1][4]. Some researchers have questioned whether the flaw is practically exploitable for code execution beyond specific builds, and Konfirmity, as cited by Sysdig, reports that the fix commit described the bug as low severity [11]. The commit message itself

reads only "Detect and report invalid stream earlier" [6][11]. These positions can be reconciled. Crashes are trivially reproducible, so any internet-facing service that decodes untrusted Brotli is exposed to denial of service now. Code execution has been shown on stock builds, and, as with other memory-corruption flaws, public exploits may become more reliable as they are refined.

The Eight-Month Public Patch Window

The February 27 upstream commit was publicly readable for about eight months before disclosure [12]. Attackers who watch security-relevant commits in widely used libraries can, in principle, diff a fix and build an exploit long before defenders receive an advisory. Nothing in the reviewed sources indicates that this happened here. Envoy and Istio shipping fixes in early October without naming the flaw would also have given careful observers a signal [5]. The pattern resembles other recent cases, such as the AnyDesk Linux fix examined in CSA Labs' AnyPwn research note, where a fix existed well before advisory-driven patch programs had any reason to prioritize it [16]. It suggests that triage based only on CVE publication can lag real exposure for embedded components.

Inventory Is the Hard Part

The main practical challenge is finding every copy of the decoder. Downstream projects may backport the fix without changing the upstream version string, so version matching can produce false positives and false negatives alike [6]. Running processes keep the flawed decoder until they restart, and container images must be rebuilt and redeployed, not just patched on the host [5]. Several vendors and researchers have released tooling to help. These include JFrog's open-source scanner, which looks for decoder references and decompression call paths in JavaScript, Python, and native ELF binaries, and runtime-focused tooling such as Sysdig's, which compares detected Node.js versions against the fixed releases [6][11][14]. JFrog cautions that a scanner match does not by itself prove remote exploitability [6].

Recommendations

Immediate Actions

Given the KEV listing and demonstrated code execution on stock builds, organizations should treat CVE-2026-111497 as an emergency patch for any internet-facing service or outbound-fetching workload that decodes Brotli. Upgrade Node.js to 22.23.4, 24.22.1, or 26.12.1 [2]. Upgrade Bun to 1.4.4 [5]. Update .NET runtimes and SDKs to the patched builds listed in Microsoft's advisory [8]. Upgrade the Python

`brtli` and `brtlicffi` packages to 1.2.1 and 1.2.1.0, and update Envoy, Istio, PHP for Windows, and operating-system Brotli packages to fixed releases [5][10]. Workloads on end-of-life Node.js lines 20, 23, and 25 have no upstream fix and should be moved to a supported line [5].

After upgrading, restart every affected process, and rebuild and redeploy container images and self-contained or NativeAOT .NET applications. Host-level updates do not replace bundled or statically linked decoders [5][8][13]. Where patching must wait, stop decompressing untrusted Brotli data. That means removing `br` from ASP.NET Core `DecompressionProviders` and from `HttpClient AutomaticDecompression` flags, disabling the Brotli decompressor filter in Envoy and Istio in both directions, and turning off Brotli decoding in HTTP clients. Blocking inbound `Content-Encoding: br` at the edge alone is not sufficient [5][8]. Microsoft warns against adding decompression configuration that is not already present, because doing so can introduce new denial-of-service risk [8].

Short-Term Mitigations

Over the following weeks, security teams should build a decoder-level inventory instead of relying on package names. That means scanning source trees, lockfiles, container images, and native binaries for embedded Brotli. Each match should be checked against the vendor's statement of whether the fix is included, since backports make version strings unreliable [6]. Teams should map which workloads fetch attacker-influenced URLs, including AI agents with browsing or retrieval tools, webhook processors, and preview services. These are the outbound exposures that edge controls miss [9].

Detection should focus on the effects of exploitation, because runtime sensors typically cannot observe the overflow itself [11]. Useful signals include repeated crashes of Node.js or .NET worker processes, especially when followed by unexpected program execution in the same workload. Shell spawns from web-facing processes, unexpected outbound connections, and unusually large Brotli-encoded request bodies are also worth alerting on [7][11]. Indicators published by ELLIO can seed short-term threat hunting [7].

Strategic Considerations

This incident shows that compression, parsing, and serialization libraries are shared attack surface across ecosystems. Software composition analysis that stops at the language package manager is likely to miss them. Organizations should require vendors to disclose embedded native components, and should push for software bills of materials that capture vendored C libraries inside runtimes and binaries. Vulnerability

management programs should also learn to track upstream fix commits and quiet downstream releases for critical embedded components, rather than waiting only for CVE publication. The eight-month gap between this fix and its disclosure shows how long that blind spot can last [12].

The finding credit to OpenAI Codex Security [1][2] is consistent with CSA's earlier observation that AI-assisted analysis is reaching mature, heavily deployed C code in foundational libraries [15][18]. Organizations should expect more disclosures of this kind, each touching many products at once, and should plan remediation capacity, rebuild pipelines, and restart procedures accordingly. Defenders can also adopt the same class of tools to audit the native dependencies their own products bundle.

CSA Resource Alignment

CSA's research note *Project Glasswing: AI Discovery Outpaces Open Source Patching Capacity* is the most directly relevant prior work [15]. It documents a structural gap between machine-speed vulnerability discovery and the capacity of open-source maintainers to ship fixes and of organizations to deploy them. CVE-2026-111497 is a concrete instance of that dynamic. An AI-assisted security tool found a memory-safety flaw in a foundational library, and the hard work was not the upstream fix but the downstream propagation through runtimes, proxies, language bindings, and container images. That paper's recommendations on prioritizing remediation capacity and tracking upstream fixes apply directly to Brotli's multi-ecosystem spread. CSA Labs' *AI-Accelerated Vulnerability Discovery and the Patch Debt Crisis* examines the same dynamic from the remediation side, and its analysis of accumulating patch debt helps explain why a single upstream fix can take months to reach deployed runtimes and container images [18].

CSA Labs' research note *AnyPwn: AnyDesk Linux Fix Described as a Crash Gains Public Exploit* examines a closely related pattern [16]. In that case, a security fix shipped months before advisory channels gave defenders a reason to prioritize it, leaving advisory-driven patch programs behind real exposure. Brotli's February 2026 upstream commit and the quiet Envoy and Istio releases follow the same shape at a larger scale. That note's guidance on monitoring vendor release notes and upstream changes for critical components applies directly to embedded libraries such as Brotli.

CSA's AI Controls Matrix (AICM) v1.1 provides the control framework for operationalizing these recommendations [17]. Its Threat and Vulnerability Management (TVM) domain covers vulnerability prioritization and remediation timelines, which organizations can apply to KEV-listed flaws such as this one. Its Application and Interface Security (AIS) domain addresses secure handling of untrusted input in application and API layers, a category that includes decompression of attacker-supplied data. Its Supply Chain Management, Transparency, and Accountability (STA) domain supports requiring vendors to

disclose embedded components. For organizations running AI agents and retrieval systems, the AICM's coverage of AI application providers and orchestrated service providers extends these controls to the outbound-fetch exposure described above.

References

- [1] CVE Program / Google. "[CVE-2026-111497: Out-of-Bounds Write in Google Brotli Decoder](#)." CVE.org, October 22, 2026 (updated October 28, 2026).
- [2] Node.js Project. "[Wednesday, October 21, 2026 Security Releases](#)." Node.js, October 21, 2026.
- [3] Google. "[Brotli v1.2.1 Release](#)." GitHub, October 21, 2026.
- [4] CISA. "[Known Exploited Vulnerabilities Catalog: CVE-2026-111497](#)." Cybersecurity and Infrastructure Security Agency, October 27, 2026.
- [5] Tenable. "[Brotli2Shell: Frequently Asked Questions for Google Brotli Decoder Vulnerability \(CVE-2026-111497\)](#)." Tenable, October 26, 2026 (updated October 27, 2026).
- [6] Engelberg, A., Tsalolikhin, E., and Ouzan, O. "[Brotli2Shell: Beyond Node\(2Shell\) and HTTP\(2Shell\)](#)." JFrog Security Research, October 25, 2026.
- [7] ELLIO. "[Node2Shell \(CVE-2026-111497\): RCE on Stock Node.js Builds](#)." ELLIO, October 25, 2026 (updated October 26, 2026).
- [8] Microsoft. "[.NET Brotli Decoder Out-of-bounds Write Vulnerability \(CVE-2026-111497\)](#)." dotnet/announcements, October 25, 2026.
- [9] Aikido Security. "[Node2Shell Brotli Vuln \(CVE-2026-111497\) Evolves to HTTP2Shell](#)." Aikido, October 23, 2026 (updated October 27, 2026).
- [10] GitHub. "[GHSA-9v8h-qggm-4m35: Out-of-bounds Write in the Brotli Decoder Also Affects the PyPI Brotli Package](#)." GitHub Advisory Database, October 22, 2026.
- [11] Sysdig. "[Node2Shell \(CVE-2026-111497\): Detecting the Node.js Brotli RCE](#)." Sysdig, October 24, 2026.
- [12] Help Net Security. "[Brotli Bug Reaches Far Beyond Node.js \(CVE-2026-111497\)](#)." Help Net Security, October 27, 2026 (updated October 28, 2026).
- [13] Aikido Security. "[Node2Shell: Node.js Brotli Vulnerability Allows Remote Code Execution \(CVE-2026-111497\) and the Fix](#)." Aikido, October 22, 2026 (updated October 27, 2026).
- [14] JFrog. "[brotli2shell cve 2026 111497 Vulnerability Scanner](#)." GitHub, October 2026.

- [15] Cloud Security Alliance. "[Project Glasswing: AI Discovery Outpaces Open Source Patching Capacity.](#)" CSA, June 2026.
- [16] Cloud Security Alliance. "[AnyPwn: AnyDesk Linux Fix Described as a Crash Gains Public Exploit.](#)" CSA Labs, October 2026.
- [17] Cloud Security Alliance. "[AI Controls Matrix \(AICM\) v1.1.](#)" CSA, June 2026.
- [18] Cloud Security Alliance. "[AI-Accelerated Vulnerability Discovery and the Patch Debt Crisis.](#)" CSA Labs, June 17, 2026.
- [19] Node.js Project. "[nodejs/node: deps/brotli.](#)" GitHub, accessed October 28, 2026.