

FACHARTIKEL · IT-SECURITY

PHP2Shell

Zwei kritische Lücken im PHP-Kern — Technik, Funktionsweise, Hintergrund und Relevanz

CVE-2026-109618 · CVE-2026-109619 · CVSS v4.0 9.2

Stand: 22. Oktober 2026

Ulrich B. Boddenberg

IT-Consulting · Software Engineering · Technologieseminare
www.boddenberg.de

Executive Summary

Normalerweise läuft eine PHP-Sicherheitswarnung so ab: Irgendein Plugin mit drei Downloads im Monat hat eine Injection-Lücke, der Administrator aktualisiert und geht wieder Kaffee holen. Diesmal nicht. Diesmal sitzt das Problem nicht im Plugin, nicht im Framework und nicht im schlampigen Code von irgendwann, sondern **im PHP-Interpreter selbst**. Eine nackte „Hello World“-Seite auf einer verwundbaren Version reicht als Angriffsfläche.

Unter dem Namen **PHP2Shell** laufen zwei Schwachstellen im PHP-Kern, beide mit **CVSS v4.0 9.2** bewertet, beide aus der Ferne, ohne Anmeldung und ohne Zutun eines Benutzers ausnutzbar. Betroffen ist praktisch jede PHP-Installation, die Webanfragen bedient, über nahezu alle Versionen seit PHP 5 hinweg. Weil die technischen Details einen Tag vor dem geplanten Patch durchgesickert sind, zog das PHP-Projekt den Release vor. Seit Mittwochnachmittag beobachten mehrere Sicherheitsfirmen aktive Ausnutzungsversuche.

Das Bundesamt für Sicherheit in der Informationstechnik hat Warnstufe „3 / Orange“ ausgerufen; einzelne Kommunen und Behörden haben Online-Dienste vorsorglich abgeschaltet. Für Betreiber gilt eine einzige belastbare Maßnahme: **Die PHP-Laufzeit aktualisieren**. Konfigurationstricks mildern einzelne Angriffswege ab, dichten aber nichts vollständig ab.

Kernaussage

Es gibt genau eine vollständige Gegenmaßnahme: Update der PHP-Laufzeit auf 8.5.12, 8.4.27, 8.3.36 oder 8.2.35 und anschließend alle Worker neu starten. Wer eine ältere Version als 8.2 fährt, hat ein Zusatzproblem, dazu später mehr.

Ausgangssituation und Einordnung

PHP treibt nach wie vor einen gewaltigen Teil des Webs an, vom kleinen Vereins-CMS bis zum Shop mit Millionenumsatz. Genau das macht eine Lücke im Interpreter so unangenehm: Sie ist nicht an ein bestimmtes Produkt gebunden. Wer WordPress, Laravel, Shopware, Nextcloud, TYPO3 oder eine selbstgebaute Anwendung betreibt, sitzt auf derselben Laufzeit und damit im selben Risiko.

Der entscheidende Punkt, den man intern immer wieder erklären muss: **Dein Anwendungscode kann makellos sein und du bist trotzdem verwundbar**. Die Lücken liegen unterhalb der Anwendung. Ein Angreifer braucht keine verwundbare Funktion in deinem Code, er braucht nur einen verwundbaren Interpreter, der eine Webanfrage beantwortet.

Faktenkasten: PHP2Shell auf einen Blick

- **CVE-2026-109618** — „RCE via OOM corruption“, GHSA-w8hw-w9fx-j79w
- **CVE-2026-109619** — „RCE via stale global state“, GHSA-6wmf-4rv5-p3rp
- Bewertung jeweils **CVSS v4.0 9.2 (kritisch)**; Vektor AV:N / AC:H / AT:P / PR:N / UI:N / VC:H / VI:H / VA:H

- Schwächenklassen u. a. CWE-416 (Use after free), CWE-755, CWE-824, CWE-1325, CWE-1419
- Betroffen: alle gängigen SAPIs, die Web bedienen (mod_php, PHP-FPM, php-cgi, FrankenPHP, LiteSpeed/LSAPI), Versionen ab PHP 5
- Behoben in **8.5.12**, **8.4.27**, **8.3.36**, **8.2.35**; Quellcode-Backports für 7.4, 8.0, 8.1
- Status: Ausnutzungsversuche beobachtet, bis Redaktionsschluss kein öffentlich bestätigter erfolgreicher Einbruch, nicht im CISA-KEV-Katalog

Technischer Hintergrund: warum das funktioniert

Beide Lücken drehen sich um dasselbe Grundthema: PHP verwaltet pro Anfrage einen eigenen Speicherbereich, und an zwei Stellen gerät dieser Bereich in einen Zustand, mit dem der Interpreter nicht sauber umgeht. Die eigentliche Ursache ist jeweils unspektakulär, die Folge ist Codeausführung. Im Folgenden ohne Exploit-Details, aber mit genug Mechanik, damit die Warnungen Sinn ergeben.

CVE-2026-109618: longjmp beim Speicherüberlauf

Wenn PHP der Speicher ausgeht, nutzt der Interpreter intern einen `longjmp()`, um die aktuelle Verarbeitung abubrechen und aus dem Kontext herauszuspringen. Das ist ein grober Sprung, der nicht ordentlich aufräumt. Dumm nur, dass der Speicher einer Anfrage in bestimmten Momenten schon erreichbar ist, bevor er vollständig initialisiert wurde. Schlägt eine Speicheranforderung genau dann fehl, springt PHP heraus und lässt den Anfragespeicher in einem ungültigen Zustand zurück. Die Aufräumroutine am Ende der Anfrage liest diesen Zustand dann und hält ihn für gültig.

Der Clou steckt im Weg dorthin: Mit einer geschickt präparierten Multipart-Anfrage lässt sich der Speicher gezielt zum Überlaufen bringen, ohne die auf den ersten Blick greifenden PHP-Grenzen zu verletzen. Eine Anfrage ohne brauchbar deklarierte Länge rutscht an der ersten `post_max_size`-Prüfung vorbei; eine Multipart-Sektion darf beliebig viele Header tragen, deren interne Darstellung ein Vielfaches ihrer Größe auf der Leitung frisst; leere Upload-Felder zählen nicht gegen `max_file_uploads`, erzeugen aber trotzdem Metadaten. So füllt eine formal harmlose Anfrage den `memory_limit`.

CVE-2026-109618: Speicher alle, mitten im Aufbau

Ablauf des Angriffs über eine präparierte Multipart-Anfrage, getrennt nach Ausführungsmodell

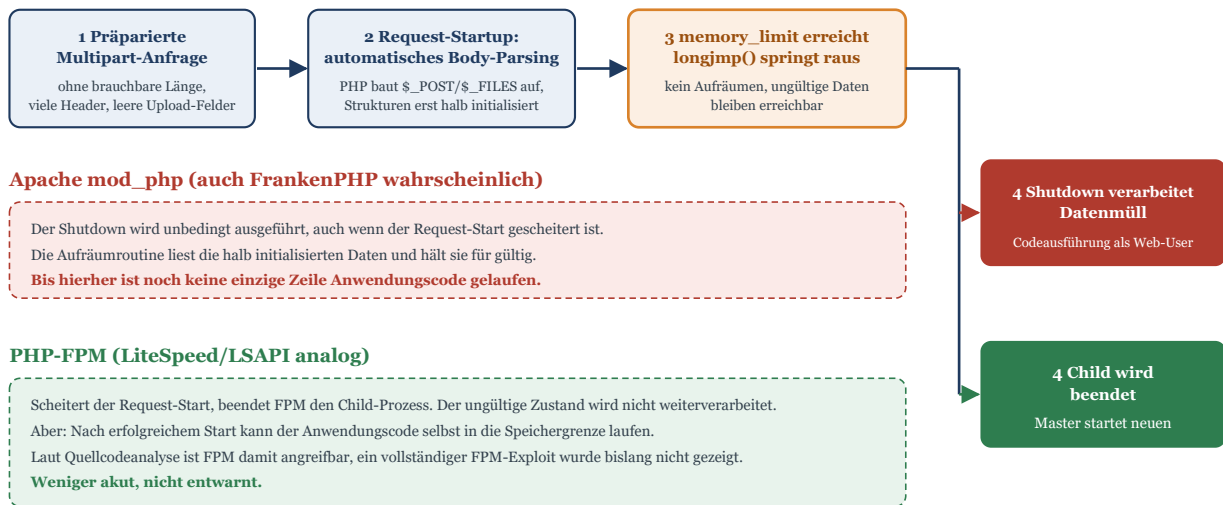


Abbildung 1: Ablauf von CVE-2026-109618. Entscheidend ist die Verzweigung nach Ausführungsmodell: `mod_php` verarbeitet den Datenmüll im Shutdown, `FPM` beendet den Child.

Die Reaktion hängt stark vom Ausführungsmodell ab. **Apache mod_php** ruft die Shutdown-Routine bedingungslos auf, selbst wenn der Request-Start gescheitert ist, und verarbeitet dabei den ungültigen Zustand, bevor auch nur eine Zeile Anwendungscode läuft. Das ist der bestätigte Weg zur Codeausführung. **PHP-FPM** beendet den Child-Prozess bei gescheitertem Start und zieht ihm damit die Grundlage weg, bleibt aber laut Quellcodeanalyse grundsätzlich angreifbar, falls der Anwendungscode nach erfolgreichem Start selbst in die Speichergrenze läuft. FrankenPHP verhält sich wie `mod_php`, LiteSpeed/LSAPI wie `FPM`.

Was der Patch ändert

Der Fix entfernt den `longjmp()` bei Speichermangel. An seine Stelle tritt ein verzögerter, geordneter „soft OOM“ am nächsten sicheren Punkt der virtuellen Maschine und, falls der Speicher eine zweite, höhere Schwelle reißt, ein harter Abbruch per `_exit()`. Die Shutdown-Routine sieht keinen ungültigen Zustand mehr. Nebenwirkung: Es gibt eine neue harte Speichergrenze, prüfe nach dem Update `memory_limit` und `post_max_size` gegen den Bedarf deiner Anwendung.

CVE-2026-109619: der Zombie-Zustand aus dem letzten Request

Die zweite Lücke ist subtiler und, ehrlich gesagt, die unangenehmere. PHP-Worker sind langlebig: Ein Prozess bedient nacheinander viele Anfragen. Dabei hält der Worker globalen Zustand, etwa Fehler-, Session-, Objekt- und Extension-Informationen, die auf Speicher zeigen, der eigentlich nur einer einzigen Anfrage gehört. Dieser Zustand muss zurückgesetzt werden, bevor der Anfragespeicher freigegeben wird. Passiert das nicht, zeigt ein globaler Zeiger ins Leere, und die nächste Anfrage greift darauf zu.

Für einen funktionierenden Angriff braucht es zwei Rollen. Der **Producer** ist eine Operation, die anfrage-eigenen Zustand global ablegt und dann durch einen Fatal Error, einen Speicherüberlauf oder einen Timeout abgebrochen wird, bevor der passende Reset läuft. Der **Consumer** ist eine

Operation der nächsten Anfrage im selben Worker, die diesen veralteten Zustand liest, bevor die normale Initialisierung ihn ersetzt, das Ergebnis ist ein Use-after-free. Bei geschickt geformtem Heap-Layout wird daraus Codeausführung.

CVE-2026-109619: Der Zombie aus dem letzten Request

Zwei Anfragen, ein Worker-Prozess: Producer hinterlässt einen Zeiger ins Leere, Consumer greift zu

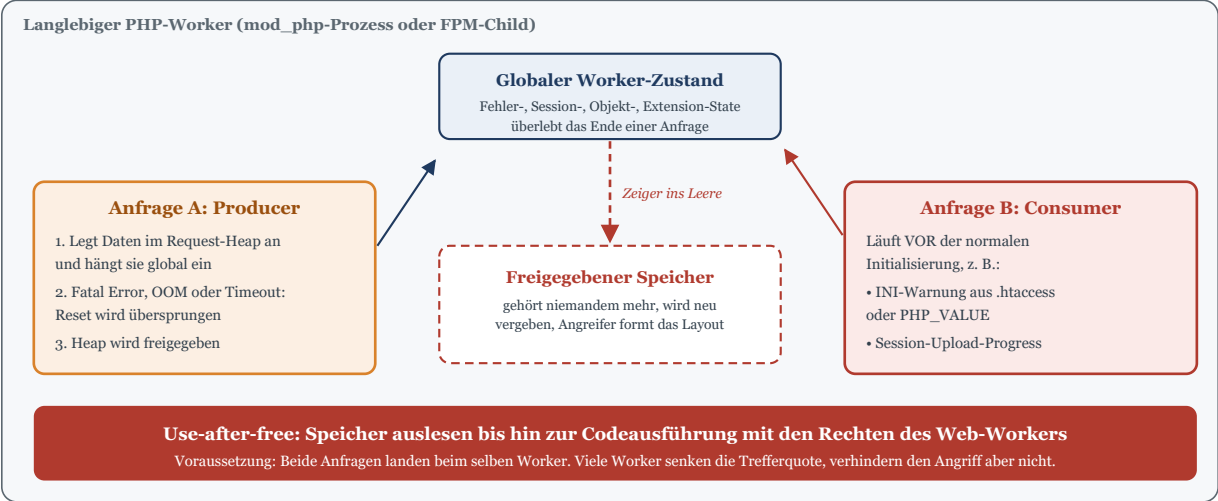


Abbildung 2: Producer-Consumer-Prinzip bei CVE-2026-109619. Der kritische Punkt ist, dass der Consumer vor der normalen Request-Initialisierung läuft.

Zwei Consumer-Wege sind bestätigt. Erstens **Diagnosemeldungen aus früh angewendeter Konfiguration**: Eine Warnung beim Anwenden ungültiger oder veralteter INI-Werte, etwa aus einer `.htaccess` unter `mod_php` oder aus `PHP_VALUE`-Parametern an FPM, greift auf den veralteten Fehlerzustand zu, noch bevor PHP entscheidet, ob die Warnung überhaupt angezeigt oder geloggt wird. `error_reporting, @` oder abgeschaltetes `display_errors` helfen deshalb nicht. Zweitens **Session Upload Progress**: Diese Funktion erreicht die Session-Verarbeitung vor der normalen Initialisierung, ist standardmäßig aktiv und braucht kein `session_start()` im Anwendungscode.

Der unangenehme Teil

Für diese Lücke gibt es keine generische Konfigurations-Gegenmaßnahme, die alle Wege abdeckt. Das Zurücksetzen des globalen Zustands lässt sich sauber nur im Code erledigen, und genau das macht der Patch. Jede Abschaltung einzelner Features (Session Upload Progress, auf PHP 8.5 zusätzlich `fatal_error_backtraces`) schließt nur den jeweils bestätigten Weg, nicht die ganze Klasse.

Analysebefunde

Die folgende Einstufung fasst zusammen, wie akut die Lage je nach Betriebsart ist. Sie ersetzt keine Inventur der eigenen Systeme, sondern hilft bei der Priorisierung.

Stufe	Befund	Betrifft
Kritisch	Unauthentifizierte Codeausführung über eine einzige Anfrage, bevor Anwendungscode läuft	Apache mit mod_php, verwundbare PHP-Version, Standardkonfiguration

Stufe	Befund	Betrifft
Kritisch	Codeausführung über den Producer-Consumer-Weg (CVE-2026-109619), validiert auf mod_php und FPM	mod_php und PHP-FPM mit aktivem Session Upload Progress
Kritisch	Keine Sicherheitsupdates mehr vom PHP-Projekt verfügbar	PHP 8.1 und älter (nur Backports über Distribution/Vendor)
Mittel	Grundsätzlich angreifbar laut Quellcodeanalyse, vollständiger Exploit bislang nicht gezeigt	PHP-FPM, LiteSpeed/LSAPI bei CVE-2026-109618
Mittel	Auslesen von Worker-Speicher (Passwörter, Tokens, Schlüssel) auch ohne Codeausführung	alle verwundbaren Installationen
Niedrig	Reines CLI oder Nicht-Web-Einsatz ohne eingehende HTTP-Anfragen	PHP ohne Web-SAPI

Hinweis zur Versionsnummer

Verlasse dich nicht blind auf die ausgegebene Versionsnummer. Linux-Distributionen und LTS-Vendoren haben die Patches vorab erhalten und liefern den Fix oft unter einer alt aussehenden Versionsnummer („Backport“). Maßgeblich ist der Changelog des Pakets mit Nennung der CVE, nicht der nackte Versionsstring.

Aktionsplan Click-by-Click

Reihenfolge für den Ernstfall. Wer mehrere Server betreibt, arbeitet die Punkte pro System ab, am besten von außen erreichbar zuerst.

Maßnahme 1 — Inventur (Priorität: hoch)

1. Alle Systeme auflisten, die PHP für Webanfragen ausführen, inklusive Containern und Images.
2. Je System die Betriebsart feststellen: mod_php, PHP-FPM, php-cgi, FrankenPHP oder LiteSpeed.
3. Die tatsächlich laufende PHP-Version ermitteln, nicht die vermutete.

Erfolgskriterium: vollständige Liste mit System, SAPI und Version. **Rollback:** entfällt (nur Lesezugriff).

Maßnahme 2 — Aktualisieren (Priorität: kritisch)

4. PHP-Laufzeit auf 8.5.12, 8.4.27, 8.3.36 bzw. 8.2.35 bringen, über Distributionspaket, Hoster oder Container-Image.
5. Danach Apache, PHP-FPM und alle langlebigen PHP-Worker neu starten. Ein Update der Anwendung (WordPress, Laravel) aktualisiert die Laufzeit darunter nicht.
6. Für PHP 8.1 und älter: Backport des Vendors einspielen oder, besser, auf eine unterstützte Version heben.

Erfolgskriterium: gepatchte Version läuft, Worker neu gestartet, Changelog nennt beide CVE.

Rollback: vorheriges Paket bzw. Image zurückspielen; vor dem Update Snapshot ziehen.

Achtung rpms.remirepo.net

Das PHP-Projekt rät Nutzern des freiwillig gepflegten Repos rpms.remirepo.net ausdrücklich, auf Distributionspakete zu wechseln oder aus dem Quellcode zu bauen, um sofort patchen zu können. Dort lag der Fix am Release-Tag nicht vor.

Maßnahme 3 — Übergangsmaßnahmen, falls sofortiges Update unmöglich (Priorität: mittel)

Diese Schritte reduzieren die Angriffsfläche einzelner Wege. Sie machen ein System **nicht** sicher und ersetzen das Update nicht.

- Gegen CVE-2026-109618: `memory_limit=-1` administrativ erzwingen (etwa per `php_admin_value` in der FPM-Pool- oder Apache-Konfiguration), sodass keine Anwendung eine endliche Grenze zurücksetzt. `post_max_size` niedrig halten. **Nicht** `RLIMIT_AS` oder `RLIMIT_DATA` verwenden, die lösen den verwundbaren Pfad gerade aus.
- Request-Größe, Feldzahl, Multipart-Teile, Header- und Verschachtelungstiefe am Webserver oder Proxy begrenzen, auch für Anfragen ohne brauchbare `Content-Length`.
- Gegen CVE-2026-109619: `session.upload_progress.enabled=0` administrativ setzen und jeden Worker neu starten. Auf PHP 8.5 zusätzlich `fatal_error_backtraces=0`.
- Veraltete oder ungültige INI-Direktiven entfernen, die früh Diagnosemeldungen erzeugen; Konfiguration bevorzugt über `php.ini`, FPM-Pool oder Serverkonfiguration beim Start statt per Request-Override.

Erfolgskriterium: Maßnahmen administrativ gesetzt und per Worker-Neustart aktiv. **Rollback:** Einstellungen zurücknehmen, sobald das Update eingespielt ist.

Maßnahme 4 — Nach dem Patch prüfen (Priorität: mittel)

7. `memory_limit` und `post_max_size` gegen den Anwendungsbedarf gegenprüfen, wegen der neuen harten Speichergrenze.
8. Falls ein früherer Einbruch im Raum steht: Logs auf ungewöhnliche Multipart-Anfragen mit anschließenden Speichergrenz-Fehlern, gehäufte HTTP-500, auffällig große PHPSESSID-Cookies und unerklärliche Worker-Neustarts durchsehen.

Erfolgskriterium: Konfiguration passt, Logs gesichtet. **Rollback:** entfällt.

Hintergrund: ein geplatzter Zeitplan und die Rolle der KI

Die Offenlegung war koordiniert und für Donnerstag, den 22. Oktober geplant. Dann ging einiges schief. Am Dienstag tauchten die beiden CVE-Nummern samt Termin in einem Forenbeitrag auf, gestützt nur auf einen vertraulichen Tipp. Das PHP-Sicherheitsteam reagierte noch am Abend mit einer Vorwarnung und zog den Release auf Mittwoch, 16:00 Uhr MESZ, vor. In der Nacht und am Morgen kam es dicker: Ein Proof-of-Concept-Video geriet versehentlich an die Öffentlichkeit, und rund sechs Stunden vor dem Patch wurden bereits gepatchte Container-Images publik, an denen sich der Fix studieren ließ, während alle anderen noch warteten.

Wie aus einer geordneten Offenlegung ein Wettrennen wurde

Zeitangaben in deutscher Sommerzeit (MESZ), 20. bis 22. Oktober 2026

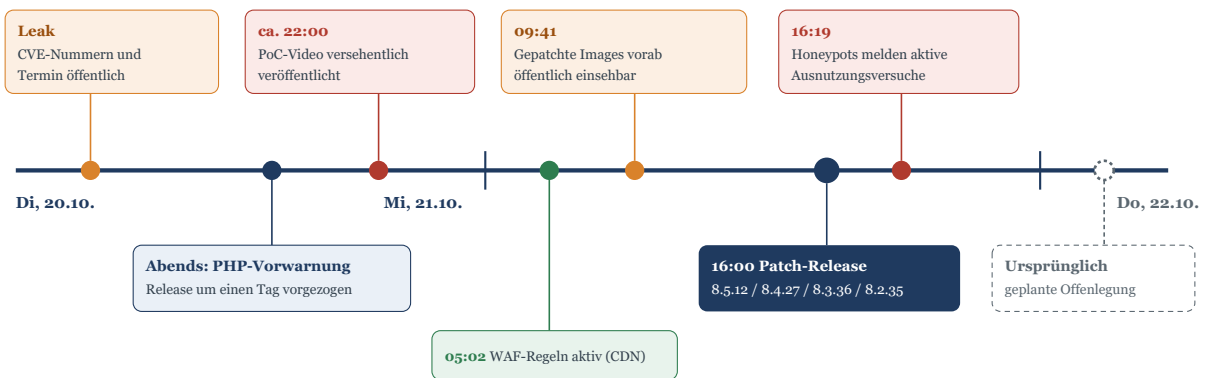


Abbildung 3: Zeitleiste vom Leak bis zu den ersten Ausnutzungsversuchen. Die geplante Offenlegung wäre erst einen Tag später gewesen.

Bemerkenswert ist die Vorgeschichte. Laut PHP Foundation stieß ein Kernentwickler bereits bei einem externen Audit 2024 auf erste Anzeichen der OOM-Lücke, ein konkreter Weg zur Codeausführung wurde damals aber nicht gefunden, und konkurrierende Fixes drohten jeweils Kompatibilität zu brechen oder neue Probleme zu öffnen. Die Arbeit stockte monatelang. Die Foundation nennt das selbstkritisch ein „strukturelles Versagen“ ihres Entscheidungsprozesses und ihrer Ressourcen.

Was 2024 nicht gelang, gelang 2026, und zwar mit KI-Unterstützung. Die Forscherteams berichten, KI-Modelle hätten geholfen, die langjährig bekannten Schwächen in funktionierende Belege zu überführen, eine Person habe in rund einer Woche geschafft, wofür ein Team sonst Monate gebraucht hätte. Wichtig für die nüchterne Einordnung: Die Modelle sind hier Werkzeug, nicht Zauberei. Calif betont, dass das Wissen um die zwanzig Jahre alte Code-Historie den Ausschlag gab und aktuelle Modelle solche obskuren Ausgangspunkte nicht zuverlässig von selbst finden.

Was das für die Praxis heißt

Die eigentliche Neuigkeit ist nicht „KI hackt PHP“, sondern die veränderte Ökonomie: Wenn sich der Spezialaufwand pro Ziel senken lässt, verschiebt sich das Gleichgewicht zwischen Entdeckung und Ausnutzung. Für Betreiber heißt das ganz praktisch: Das Zeitfenster zwischen Offenlegung und breiter Ausnutzung wird eher kürzer. Schnelles, geprobt Patchen ist kein Hygienethema mehr, sondern Betriebsvoraussetzung.

Risiken

- **Datenabfluss ohne Codeausführung.** Schon das Auslesen von Worker-Speicher kann Passwörter, Authentifizierungs-Tokens, API-Schlüssel und Zertifikate anderer Anfragen preisgeben. Kompromittierte Geheimnisse bleiben auch nach dem Patch kompromittiert und gehören rotiert.
- **Betriebsunterbrechung durch Vorsichtsmaßnahmen.** Einzelne Organisationen haben Dienste abgeschaltet, nicht wegen eines Einbruchs, sondern um keine Angriffsfläche zu bieten. Das ist eine legitime, aber teure Option.

- **Nebenwirkung der neuen Speichergrenze.** Der Patch führt eine harte Grenze ein, die beim Erreichen den Prozess beendet und Shutdown-Handler nicht mehr laufen lässt. Anwendungen mit hohem Speicherbedarf sollten `memory_limit` und `post_max_size` nach dem Update prüfen.
- **Scheinsicherheit durch Teil-Mitigationen.** Wer nur Session Upload Progress abschaltet und sich in Sicherheit wiegt, übersieht die übrigen Wege. Nur das Update ist vollständig.
- **Alt-Versionen ohne Pflege.** PHP 8.1 und älter bekommen keine regulären Updates mehr, PHP 8.2 erreicht am 31.12.2026 das Lebensende. Solche Systeme sind das Hauptrisiko und gehören zuerst gehoben.

Offene Punkte

Punkt	Stand	Nächster Schritt
Bestätigter erfolgreicher Einbruch	Bis Redaktionsschluss öffentlich nicht dokumentiert; Ausnutzungsversuche laufen	Lageberichte von PHP Foundation, BSI und Sicherheitsfirmen weiter verfolgen
Vollständiger FPM-Exploit für CVE-2026-109618	Laut Quellcodeanalyse möglich, nicht demonstriert	FPM-Systeme trotzdem priorisiert patchen
Aufnahme in CISA-KEV-Katalog	Zum Zeitpunkt der ersten Advisories nicht gelistet	Katalog beobachten, ändert nichts an der Patch-Dringlichkeit
Weitere Zustandsfamilien bei CVE-2026-109619	Patch setzt breiteren Zustand zurück, Reichweite anwendungsabhängig	Nur durch Update vollständig abgedeckt

Anhang

Betroffene und behobene Versionen

PHP-Zweig	Verwundbar	Erste sichere Version
8.5	unter 8.5.12	8.5.12
8.4	unter 8.4.27	8.4.27
8.3	unter 8.3.36	8.3.36
8.2	unter 8.2.35	8.2.35 (EOL 31.12.2026)
7.4 / 8.0 / 8.1	betroffen	nur Quellcode-Backports, kein offizielles Release

Primärquellen

Advisory des PHP-Projekts: php.net/archive/2026.php (21.10.2026)

GitHub Security Advisory CVE-2026-109618: [GHSA-w8hw-w9fx-j79w](#)

GitHub Security Advisory CVE-2026-109619: [GHSA-6wmf-4rv5-p3rp](#)

PHP Foundation zur Offenlegung: thephp.foundation/blog (21.10.2026)

NVD-Einträge: CVE-2026-109618 · CVE-2026-109619