PHP-Sicherheitslücken: Was ist PHP2Shell?

PHP-Sicherheitslücken: Was ist PHP2Shell?

Stand: 21. Oktober 2026

Für wen ist diese FAQ?

Diese FAQ richtet sich in erster Linie an Sie, wenn Sie einen vServer, StorageServer, RootServer oder DedicatedServer bei uns gebucht haben. Auf diesen Servern verwalten Sie PHP selbst und sind auch für die Sicherheitsupdates verantwortlich.

Worum geht es?

PHP2Shell ist der inoffizielle Name für zwei kritische Schwachstellen im PHP-Interpreter mit den Kennungen CVE-2026-109618 und CVE-2026-109619. Das Sicherheitsteam des PHP-Projekts bewertet beide mit 9,2 von 10 Punkten (CVSS v4.0). Entdeckt haben sie Sicherheitsforscher von Calif, Wiz Research und OpenAI. Der Name lehnt sich an die Log4j-Lücke „Log4Shell“ aus dem Jahr 2021 an.

Am 20. Oktober wurden die Kennungen und erste Informationen vor dem vereinbarten Veröffentlichungstermin bekannt; kurz darauf kursierte ein inzwischen gelöschtes Video der Forscher, das einen funktionierenden Angriff zeigte. Das PHP-Projekt kündigte noch am selben Tag ein außerplanmäßiges Sicherheitsupdate an und veröffentlichte es am 21. Oktober, einen Tag früher als ursprünglich geplant. Zusammen mit dem Update hat das PHP-Projekt die technischen Einzelheiten in zwei Sicherheitshinweisen offiziell bekannt gegeben.

Die Lücken stecken in PHP als Laufzeitumgebung, nicht in einem einzelnen CMS oder Plugin. Betroffen sind deshalb grundsätzlich alle PHP-basierten Websites, etwa solche mit WordPress, Joomla, TYPO3, Shopware, Magento oder Eigenentwicklungen, aber auch andere Produkte, die PHP enthalten.

Wie funktionieren die Lücken grundsätzlich?

Bei beiden handelt es sich um Speicherfehler im Kern von PHP:

- **CVE-2026-109618** betrifft den Umgang mit Speichermangel (Out of Memory). Erreicht PHP beim Verarbeiten einer Anfrage das Speicherlimit, bricht es mitten in der Vorbereitung ab; beim anschließenden Aufräumen verwendet PHP dann unvollständig vorbereitete Daten. Bestätigte Auslöser sind präparierte Anfragen im Format multipart/form-data, das vor allem für Datei-Uploads genutzt wird. Unter mod_php lässt sich die Lücke laut Mattias Geniar auch mit normalen Formularanfragen (application/x-www-form-urlencoded) auslösen.
- **CVE-2026-109619** betrifft Zustände, die eine Anfrage zurücklässt. Nach einem fatalen Fehler, etwa wegen Speichermangels oder einer Zeitüberschreitung, können Verweise auf bereits freigegebenen Speicher übrig bleiben, die eine spätere Anfrage an denselben PHP-Prozess verwendet. Bestätigt sind Angriffswege über die Upload-Fortschrittsfunktion (session.upload_progress) und unter PHP 8.5 über die Backtraces bei fatalen Fehlern (fatal_error_backtraces).

Drei Eigenschaften machen die Lücken besonders gefährlich:

- **Unterhalb der Anwendung:** Die Fehler stecken in PHP selbst. Bei Apache mit mod_php können präparierte Anfragen den Angriff auslösen, noch bevor Ihr CMS oder Ihre Anwendung startet. Unter PHP-FPM wurde ein Angriff in Verbindung mit gängigen Vorgängen der Anwendung nachgewiesen, etwa dem Starten einer Session. Sicherheits-Plugins, Login-

© 2026 1blu AG <info@1blu.de> |
URL: https://faq.1blu.de/content/448/1012/de/php_sicherheitsluecken-was-ist-php2shell.html

Prüfungen oder Anwendungs-Updates schützen deshalb nicht.
- **Ohne Anmeldung:** Für einen Angriff sind keine Zugangsdaten nötig.
- **Standardkonfiguration:** Die betroffenen Funktionen sind in der PHP-Standardkonfiguration aktiv.

Gelingt der Angriff, kann der Angreifer eigenen Code mit den Rechten des PHP-Prozesses ausführen und typischerweise eine Shell auf dem Server öffnen. Die Forscher haben außerdem gezeigt, dass sich Speicherinhalte des PHP-Prozesses auslesen lassen, auch ohne dass Code ausgeführt wird. Darin können Passwörter, Session-Tokens oder API-Schlüssel aus Anfragen anderer Besucher stehen, unter Umständen auch private TLS-Schlüssel des Servers.

Auf weitere technische Einzelheiten verzichten wir hier bewusst. Maßgeblich sind die Sicherheitshinweise des PHP-Projekts.

Bin ich betroffen?

Ja, wenn PHP auf Ihrem Server Web-Anfragen verarbeitet und noch keine Korrektur enthält. Besonders gefährdet sind Websites, die aus dem Internet erreichbar sind. Das PHP-Projekt nennt alle Versionen vor 8.5.12, 8.4.27, 8.3.36 und 8.2.35 als betroffen und hat zusätzlich Korrekturen für PHP 7.4, 8.0 und 8.1 bereitgestellt. Laut den Entdeckern reicht der verwundbare Code bis PHP 5.4 zurück.

Betroffene PHP-Versionen und Updates des PHP-Projekts:

PHP-Version	Status beim PHP-Projekt	Gepatchte Version
8.5, 8.4	aktiv unterstützt	8.5.12 bzw. 8.4.27
8.3, 8.2	nur Sicherheitsupdates	8.3.36 bzw. 8.2.35
8.1, 8.0, 7.4	End of Life	keine (nur Quellcode-Patches)
7.3 und älter	End of Life	keine

Linux-Distributionen haben die Korrekturen vorab erhalten und liefern sie oft als eigene Paketversion aus, ohne dass sich die PHP-Versionsnummer ändert. Ubuntu etwa hat bereits am 21. Oktober aktualisierte Pakete veröffentlicht, auch für PHP 8.1 unter Ubuntu 22.04. Maßgeblich sind deshalb die Sicherheitsmeldungen Ihres Distributors.

Betroffen sind alle Betriebsarten, in denen PHP Web-Anfragen verarbeitet: PHP-FPM, Apache mit mod_php, php-cgi, FrankenPHP und LiteSpeed (lsphp). Das höchste Risiko sieht das PHP-Projekt bei mod_php. Auch PHP unter Windows ist betroffen; korrigierte Windows-Versionen stellt das PHP-Projekt auf seiner Download-Seite bereit.

Werden die Lücken bereits ausgenutzt?

Es gibt Angriffsversuche, aber bislang keinen bestätigten erfolgreichen Einbruch:

- Laut dem CTO von Cloudflare wurden am Morgen des 21. Oktober, noch vor dem Release, erste Angriffe beobachtet. Inzwischen verändern Angreifer ihre Anfragen, um die Filterregeln von Cloudflare zu umgehen.
- Die Sicherheitsfirma watchTowr registriert seit dem Nachmittag des 21. Oktober Angriffsversuche auf beide Schwachstellen.
- Die Sicherheitsfirma Huntress hatte bis zum Abend des 21. Oktober bei ihren Kunden keine Ausnutzung festgestellt. Das BSI konnte Berichte über Angriffsversuche bislang nicht verifizieren.

Die Entdecker haben inzwischen technische Analysen veröffentlicht und erfolgreiche Angriffe ohne Anmeldung vorgeführt, unter anderem auf WordPress, Drupal, Joomla, Magento und Nextcloud, aber auch auf phpMyAdmin, die Webmail-Software Roundcube und die Kundenverwaltung WHMCS. Das BSI stuft den Proof-of-Concept als öffentlich ein und rechnet, wie auch Sicherheitsforscher etwa von

© 2026 1blu AG <info@1blu.de> |

Sansec, mit einer breiten Ausnutzung in kurzer Zeit. Gehen Sie davon aus, dass ungepatchte, öffentlich erreichbare PHP-Systeme angegriffen werden.

Vorsicht: Im Netz kursieren angebliche Exploits und Prüfskripte. Gefälschte Proof-of-Concepts enthalten bei solchen Anlässen häufig Schadsoftware. Führen Sie keinen ungeprüften fremden Code auf Ihren Systemen aus.

Was ist die wichtigste Maßnahme?

Das PHP-Update so schnell wie möglich einspielen. Nach Aussage des PHP-Projekts ist das die einzige vollständige Abhilfe; alle anderen Maßnahmen verringern nur das Risiko.

Dabei sollten Sie beachten:

- Updates über die Paketquellen Ihrer Distribution oder Ihres Panels (z. B. Plesk, cPanel, DirectAdmin) beziehen. Wenn Sie zusätzliche Paketquellen wie Remi oder die Pakete von Ondřej Surý nutzen, prüfen Sie gesondert, ob dort bereits aktualisierte Pakete vorliegen. Zu Remi hatte das PHP-Projekt am 21. Oktober erklärt, dass es dort am selben Tag keine Updates gebe, und zum Wechsel auf die Pakete der Distribution geraten. Laut Mattias Geniar liefert Remi die Korrektur inzwischen doch aus; einzelne Mirrors können aber noch veraltet sein (Abhilfe: dnf clean metadata).
- Alle installierten PHP-Versionen aktualisieren, nicht nur die Standardversion. Dazu gehören auch PHP-Installationen, die Hosting-Panels teils für ihre eigene Oberfläche, Webmail oder phpMyAdmin mitbringen.
- Nach dem Update alle PHP-Prozesse neu starten; laufende Prozesse behalten sonst den alten Code im Speicher. Das betrifft PHP-FPM, Apache mit mod_php (den Dienst stoppen und wieder starten, etwa mit systemctl restart apache2 bzw. systemctl restart httpd; apachectl graceful und apachectl -k restart reichen laut Mattias Geniar nicht), LiteSpeed (lsphp-Prozesse beenden) und langlaufende Worker (z. B. Queue-Worker, Laravel Octane, Swoole, RoadRunner).
- Docker-Images mit PHP neu ziehen bzw. neu bauen und die Container neu erstellen. Ein Update des Host-Systems aktualisiert PHP in Containern nicht.
- Programme mit fest eingebautem PHP (z. B. FrankenPHP) werden nicht über das Paketsystem aktualisiert; hier ist ein neues Release des jeweiligen Projekts nötig. FrankenPHP enthält die Korrektur ab Version 1.13.2. Laravel Octane lädt ein eigenes FrankenPHP-Programm herunter und akzeptiert dabei auch die ungepatchten Versionen 1.13.0 und 1.13.1. Löschen Sie deshalb die Datei frankenphp im Projektverzeichnis und starten Sie Octane neu, damit es eine aktuelle Version lädt.
- Anschließend prüfen, ob die Korrektur tatsächlich aktiv ist. php -v zeigt nur die Version auf der Kommandozeile; PHP-FPM und mod_php können eine andere Version nutzen. Prüfen Sie deshalb auch PHP-FPM selbst, etwa mit php-fpm8.3 -v unter Debian/Ubuntu oder php-fpm -v unter RHEL, bzw. bei mod_php die installierten Pakete (dpkg -l 'php*' 'libapache2-mod-php*' bzw. rpm -qa 'php*'). Verlässlich ist eine Version ab 8.5.12, 8.4.27, 8.3.36 bzw. 8.2.35. Viele Distributionen spielen die Korrektur aber ein, ohne die PHP-Versionsnummer zu ändern; dann zeigt auch php-fpm -v die alte Nummer, und nur die Paketversion ändert sich (unter Ubuntu 24.04 etwa auf 8.3.6-0ubuntu0.24.04.13). Maßgeblich ist in diesem Fall die korrigierte Paketversion aus der Sicherheitsmeldung Ihres Distributors. Zusätzlich kann ein Eintrag der CVE-Nummern im Paket-Changelog die Korrektur bestätigen (z. B. zgrep -E 'CVE-2026-10961[89]' /usr/share/doc/php*/changelog.Debian.gz unter Debian/Ubuntu oder rpm -q --changelog php-fpm | grep -E 'CVE-2026-10961[89]' unter RHEL und verwandten Systemen). Fehlt ein solcher Eintrag, heißt das aber nicht, dass die Korrektur fehlt: Manche Changelogs nennen nur die neue Upstream-Version oder keine CVE-Nummern. Ob noch Prozesse mit altem Code laufen, zeigen needrestart bzw. dnf needs-restarting.

Was ändert sich nach dem Update?

© 2026 1blu AG <info@1blu.de> |

Die Korrektur ändert, wie PHP mit dem Speicherlimit umgeht: Überschreitet ein Skript memory_limit, bricht PHP es an der nächsten sicheren Stelle ab. Zusätzlich gibt es eine harte Obergrenze oberhalb dieses Limits. Sie liegt laut Mattias Geniar standardmäßig beim Doppelten von memory_limit, höchstens aber 1 GB darüber. Erreicht ein Prozess sie, beendet PHP ihn sofort mit dem Exit-Code 137, und Shutdown-Funktionen der Anwendung laufen nicht mehr. Im PHP-FPM-Log steht dann „exited with code 137“, und nginx liefert den Fehler 502. Die zugehörige Meldung „Hard memory limit of … bytes exhausted“ schreibt PHP nicht ins PHP-Fehlerlog, sondern nur auf die Standardfehlerausgabe: Bei mod_php steht sie im Apache-Fehlerlog, bei PHP-FPM nur mit catch_workers_output = yes im FPM-Log. Verwechseln Sie diesen Abbruch nicht mit dem OOM-Killer des Kernels: Beendet dieser einen Worker, meldet PHP-FPM „exited on signal 9 (SIGKILL)“. Beobachten Sie nach dem Update deshalb die PHP- und Webserver-Logs, besonders bei speicherintensiven Skripten. Berücksichtigen Sie bei der Speicherplanung, etwa für pm.max_children, dass einzelne Prozesse kurzzeitig mehr Speicher als memory_limit belegen können.

Nehmen Sie außerdem Übergangsmaßnahmen zurück, die den Betrieb einschränken: memory_limit wieder auf einen festen Wert setzen, Sperren für Upload-Anfragen am Webserver, Proxy oder CDN aufheben, gesenkte Grenzen wie post_max_size oder max_input_vars in PHP und am Webserver bei Bedarf wieder anheben sowie pm.max_requests und fatal_error_backtraces zurücksetzen. Die Upload-Fortschrittsfunktion können Sie abgeschaltet lassen; das PHP-Projekt hält das sogar für ratsam, wenn Sie sie nicht benötigen. Eine Absicherung per SELinux oder AppArmor sollten Sie ebenfalls beibehalten.

Welche Übergangsmaßnahmen gibt es bis zum Update?

Lässt sich das Update nicht sofort einspielen, raten BSI und CERT-EU, eine vorübergehende Trennung betroffener Systeme vom Internet zu erwägen. Ist auch das nicht möglich, sollten Sie alle Maßnahmen der beiden folgenden Abschnitte gemeinsam umsetzen, soweit sie nicht als optional gekennzeichnet sind. Sie blockieren die vom PHP-Projekt bestätigten Angriffswege, aber nicht alle möglichen Varianten, und ersetzen das Update nicht. Ob sie auch den Angriffsweg über den OPcache abdecken, mit dem Wiz Research WordPress angegriffen hat, ist nicht bestätigt.

Gegen CVE-2026-109618 (Speichermangel)

- **Speicherlimit aufheben (memory_limit=-1):** Entfernt das feste Limit, auf das sich der bekannte Angriff stützt. Schlägt eine Speicheranforderung aus anderen Gründen fehl, etwa bei einem echten Speicherengpass, kann die Lücke aber weiterhin ausgelöst werden. Setzen Sie den Wert so, dass Anwendungen ihn nicht ändern können (php_admin_value), und prüfen Sie, dass keine Pool-, Verzeichnis- oder Anwendungseinstellung wieder ein Limit setzt. Unter PHP 8.5 muss außerdem max_memory_limit auf dem Standardwert -1 stehen; sonst setzt PHP memory_limit mit einer Warnung auf diesen Höchstwert. Einschränkung: Ohne Limit kann eine einzelne Anfrage den gesamten Arbeitsspeicher belegen. Begrenzen Sie den Speicher deshalb über cgroups bzw. das Container-Limit, möglichst pro Worker – nicht über RLIMIT_AS oder RLIMIT_DATA (siehe „Was hilft nicht?“). systemd stoppt standardmäßig den gesamten Dienst, wenn der Kernel wegen Speichermangels einen einzelnen Worker beendet. Setzen Sie deshalb neben MemoryMax= auch OOMPolicy=continue, etwa per systemctl edit für den PHP-FPM- bzw. Apache-Dienst.
- **Anfragegröße begrenzen:** post_max_size auf 8M setzen, wenn keine Datei-Uploads nötig sind, sonst auf 32M; upload_max_filesize muss darunter liegen. Größere Uploads sind dann nicht mehr möglich. Ergänzend am Webserver oder Proxy die Größe von Anfragen, die Zahl der Felder, Multipart-Teile und Header sowie die Verschachtelungstiefe begrenzen, auch für Anfragen ohne Längenangabe (Content-Length). In PHP selbst lassen sich dazu max_input_nesting_level (Standard: 64), max_input_vars, max_multipart_body_parts und max_file_uploads so weit senken, wie Ihre Anwendung es zulässt. Das senkt das Risiko, schließt die Lücke aber nicht.
- **Upload-Anfragen blockieren:** Benötigt Ihre Website keine Anfragen im Format

© 2026 1blu AG <info@1blu.de> |

multipart/form-data, können Sie diese am Webserver oder Proxy abweisen. Das Format nutzen Datei-Uploads, aber auch manche Formulare ohne Datei und JavaScript-Funktionen mit FormData. Die Sperre blockiert die bestätigten Angriffswege über Uploads. Weil die Upload-Fortschrittsfunktion nur bei solchen Anfragen greift, blockiert sie auch die bestätigten Angriffswege über Sessions bei CVE-2026-109619. Unter mod_php reicht sie allein nicht, weil sich CVE-2026-109618 dort laut Mattias Geniar auch mit normalen Formularanfragen auslösen lässt; den Weg über Backtraces unter PHP 8.5 deckt sie ebenfalls nicht ab. Betroffene Uploads und Formulare funktionieren dann nicht mehr; freigegebene Ausnahmen bleiben angreifbar.

Gegen CVE-2026-109619 (Restzustand zwischen Anfragen)

- **Upload-Fortschritt abschalten (session.upload_progress.enabled=0):** Blockiert die bestätigten Angriffswege über Sessions, nicht aber andere Varianten. Gegen CVE-2026-109618 hilft die Einstellung nicht. Uploads funktionieren weiter; es entfallen nur Fortschrittsanzeigen, die diese Funktion nutzen. Gängige CMS nutzen sie kaum.
- **Nur PHP 8.5 – Backtraces abschalten (fatal_error_backtraces=0):** Blockiert den bestätigten Angriffsweg über Backtraces bei fatalen Fehlern.
- **Warnungen beim Start einer Anfrage vermeiden:** Entfernen Sie ungültige oder veraltete PHP-Einstellungen, die pro Anfrage gesetzt werden (per .htaccess, in der Apache-Konfiguration oder per PHP_VALUE bzw. PHP_ADMIN_VALUE vom Webserver an PHP-FPM), zum Beispiel einen ungültigen Wert für memory_limit oder das veraltete auto_detect_line_endings=1. Solche Meldungen finden Sie, wenn Sie in der php.ini error_reporting=E_ALL und log_errors=On setzen und ein funktionierendes error_log angeben. Die Meldungen nur auszublenden (display_errors, error_reporting, @), hilft nicht.
- **Fatale Fehler vermeiden:** Beheben Sie fatale Fehler und Zeitüberschreitungen, die Angreifer gezielt auslösen können, etwa in Upload- oder Importfunktionen.
- **Optional bei PHP-FPM (pm.max_requests=1):** Jeder Worker beendet sich nach einer Anfrage, sodass kein prozesseigener Zustand auf die nächste Anfrage übergeht. Gegen Zustände im gemeinsamen Speicher, etwa im OPcache, hilft das nicht. Das kostet deutlich Leistung.

Zusätzliche Schutzschichten

- **Web Application Firewall (z. B. die Regeln von Cloudflare):** Filtert bekannte Angriffsmuster. Angreifer passen ihre Anfragen bereits an, um solche Regeln zu umgehen. Wirkt nur, wenn der Server nicht direkt, an der WAF vorbei, erreichbar ist.
- **Programmausführung durch PHP unterbinden (SELinux/AppArmor):** Verhindert den typischen letzten Schritt zur Shell, nicht aber den Speicherfehler selbst oder das Auslesen von Speicher (siehe nächste Frage).

Hinweise zur Umsetzung

- Setzen Sie die PHP-Einstellungen in der php.ini, in der Konfiguration des PHP-FPM-Pools (php_admin_value bzw. php_admin_flag) oder in der Webserver-Konfiguration, bei FrankenPHP im Caddyfile. Über php_admin_value bzw. php_admin_flag gesetzte Werte lassen sich zudem nicht durch eigene Konfigurationen im Webspace überschreiben.
- Einstellungen per .user.ini oder ini_set() greifen laut PHP-Projekt zu spät und schützen nicht.
- Starten Sie danach alle PHP-Prozesse neu und prüfen Sie den tatsächlich wirksamen Wert für jede PHP-Version und Betriebsart.
- Konkrete Konfigurationsbeispiele finden Sie in den Quellen unter Wo finde ich weitere Informationen?. Testen Sie Änderungen vor dem produktiven Einsatz.

Kann ich SELinux oder AppArmor nutzen?

Ja, als zusätzliche Schutzschicht. Das BSI empfiehlt, SELinux bzw. AppArmor zu nutzen, um die Folgen solcher Lücken zu begrenzen; auch der Sicherheitsforscher Kevin Beaumont hat den Ansatz

© 2026 1blu AG <info@1blu.de> |

gegen PHP2Shell empfohlen. SELinux (u. a. RHEL, AlmaLinux, Rocky Linux, Fedora, neue Installationen von SUSE ab Version 16) und AppArmor (u. a. Ubuntu, Debian, SUSE bis Version 15) sind Sicherheitsmodule des Linux-Kernels. Sie legen fest, was ein Prozess darf – unabhängig davon, was im Prozess selbst passiert.

Warum das hier hilft: Ein erfolgreicher Angriff endet typischerweise damit, dass der PHP-Prozess eine Shell oder ein anderes Programm startet. Darf der PHP-Prozess per Richtlinie keine fremden Programme ausführen, scheitert dieser Schritt.

Wichtig: Dass SELinux oder AppArmor aktiv ist, heißt noch nicht, dass PHP eingeschränkt ist. Die mitgelieferten Standardrichtlinien erlauben PHP in der Regel, Programme zu starten; unter Debian und Ubuntu läuft PHP-FPM meist ganz ohne AppArmor-Profil. Schutz bringt erst eine passende eigene Richtlinie. Ein strenges Profil für PHP-FPM, das das Starten anderer Programme verbietet, enthält dort das nicht standardmäßig installierte Paket apparmor-profiles. Es kann als Ausgangspunkt dienen, muss aber an Ihre Anwendung angepasst und zunächst im Protokollmodus getestet werden.

Was der Ansatz nicht leistet: Der Speicherfehler selbst bleibt bestehen, ebenso das Auslesen von Speicherinhalten. Code, der direkt im PHP-Prozess läuft, kann weiterhin alles lesen und schreiben, was PHP lesen und schreiben darf, zum Beispiel Datenbank-Zugangsdaten auslesen oder eine Hintertür als PHP-Datei im Webspace ablegen. SELinux und AppArmor begrenzen also den Schaden, ersetzen aber nicht das Update.

Grundzüge der Umsetzung:

- Prüfen, ob SELinux bzw. AppArmor aktiv ist und im erzwingenden und nicht nur im protokollierenden Modus läuft.
- Für den PHP-Prozess (PHP-FPM bzw. den Webserver bei mod_php) eine Richtlinie oder ein Profil verwenden, das das Starten von Shells und anderen Programmen verbietet. Prüfen Sie dabei, ob für PHP überhaupt eine Richtlinie greift.
- Vorher klären, ob Ihre Anwendung legitim externe Programme aufruft (z. B. Bildverarbeitung, Virenscanner, Mailversand über sendmail). Diese Aufrufe müssen gezielt erlaubt oder umgestellt werden.
- Neue Regeln zunächst im Lern- bzw. Protokollmodus testen und die Logs auswerten, bevor Sie sie scharf schalten.

Eine fertige Anleitung geben wir hier bewusst nicht, weil Richtlinien stark von Distribution, PHP-Betriebsart und Anwendung abhängen. Ausgangspunkte für Ihre eigene Recherche sind die SELinux- bzw. AppArmor-Dokumentation Ihrer Distribution sowie die unten aufgeführten Suchbegriffe.

Ergänzend lässt sich in PHP die Liste deaktivierter Funktionen (disable_functions) um Funktionen zur Programmausführung erweitern. Bei Speicherfehlern bietet das aber keinen verlässlichen Schutz und ersetzt die Absicherung auf Betriebssystemebene nicht.

Was hilft nicht?

Folgende Maßnahmen schützen allein nicht oder schaden sogar:

- **Updates von CMS, Plugins oder Themes:** Die Lücken stecken in PHP selbst.
- **Maßnahmen im Anwendungscode:** Sicherheits-Plugins, Login-Schutz oder Wartungsmodus greifen zu spät.
- **PHP-Einstellungen per .user.ini oder ini_set():** Sie werden zu spät angewendet.
- **Abschalten der Datei-Uploads (file_uploads = Off):** PHP liest Upload-Anfragen trotzdem ein und überspringt nur die Dateien.
- **Fehlermeldungen ausblenden (display_errors, error_reporting, @-Operator):** Die betroffenen Vorgänge laufen trotzdem ab.
- **Speicherbegrenzung per RLIMIT_AS oder RLIMIT_DATA (z. B. ulimit -v oder**

© 2026 1blu AG <info@1blu.de> |

LimitAS= in systemd): Sie kann genau den verwundbaren Fehlerpfad auslösen und schadet daher.
- **WAF oder CDN bei direkt erreichbarem Server:** Angreifer können den Schutz umgehen, indem sie den Server direkt ansprechen.

Was tun bei alten PHP-Versionen (8.1 und älter)?

Für PHP 8.1 und älter veröffentlicht das PHP-Projekt keine neuen Versionen mehr. Für PHP 7.4, 8.0 und 8.1 hat das PHP-Sicherheitsteam aber Quellcode-Patches bereitgestellt; sie richten sich an Betreiber, die PHP selbst kompilieren. Für noch ältere Versionen gibt es vom PHP-Projekt keine Korrektur, obwohl sie nach Angaben der Entdecker bis zurück zu PHP 5.4 betroffen sind. Einzelne Distributoren liefern aber auch dafür Updates.

Mögliche Wege:

- Prüfen, ob Ihr Distributor oder ein Anbieter erweiterter Pflege für ältere PHP-Versionen die Korrektur bereits ausliefert. Ubuntu etwa hat aktualisierte Pakete für Ubuntu 22.04 (PHP 8.1) veröffentlicht, für Ubuntu 20.04 (PHP 7.4) und 18.04 (PHP 7.2) über Ubuntu Pro (ESM), für 16.04 (PHP 7.0) und 14.04 (PHP 5.5) über dessen Legacy-Support. AlmaLinux hat auch ältere PHP-Versionen ab 7.2 korrigiert. Für Debian 11 (PHP 7.4) gibt es die Korrektur dagegen nur noch über das kostenpflichtige ELTS-Angebot von Freexian.
- Bis dahin die Übergangsmaßnahmen und SELinux/AppArmor aktiv halten.
- Die Migration auf eine unterstützte PHP-Version einplanen. Beachten Sie: Auch PHP 8.2 erhält nur noch bis zum 31. Dezember 2026 Sicherheitsupdates.
- Systeme, die sich weder aktualisieren noch absichern lassen, vom Netz nehmen.

Woran erkenne ich eine mögliche Kompromittierung?

Erste Angriffsversuche wurden bereits vor dem Release gemeldet. Prüfen Sie Ihre Systeme deshalb auch nach dem Update und sichern Sie die Logs jetzt, bevor die Logrotation sie löscht.

Mögliche Hinweise:

- PHP- oder Webserver-Prozesse (php-fpm, php-cgi, lsphp, httpd, apache2), die Shells, Interpreter wie python oder perl oder Werkzeuge wie curl, wget oder nc starten.
- Auffällig viele POST-Anfragen in kurzer Zeit, vor allem Upload-Anfragen an Pfade, die normalerweise keine Uploads erhalten, gefolgt von Fehlermeldungen zum Speicherlimit („Allowed memory size … exhausted“) oder Serien von HTTP-500-Fehlern.
- Ungewöhnlich große oder fehlerhafte PHPSESSID-Cookies in den Antworten des Servers; sie können auf das Auslesen von Speicher hindeuten. Antwort-Header stehen allerdings meist nicht im Access-Log.
- Häufige Abstürze oder unerklärliche Neustarts von PHP-Prozessen, etwa Meldungen wie „exited on signal 11 (SIGSEGV)“ im PHP-FPM-Log bzw. „exit signal Segmentation fault (11)“ im Apache-Fehlerlog.
- Unbekannte Dateien im Webspace, neue Administratorkonten im CMS, neue Cronjobs, unerwartete Weiterleitungen oder ausgehende Verbindungen.
- Bei aktivem SELinux oder AppArmor: Einträge im Audit- bzw. Kernel-Log über verweigerte Programmausführungen durch PHP.

Unauffällige Logs beweisen nicht, dass kein Angriff stattgefunden hat.

Bei konkretem Verdacht sollten Sie das System isolieren, Spuren sichern und es neu aufsetzen oder aus einem sicher sauberen Backup wiederherstellen. Ändern Sie anschließend alle Zugangsdaten und Schlüssel, auf die PHP Zugriff hatte (Datenbank, API-Schlüssel, Admin-Konten), und machen Sie bestehende Sessions ungültig. Weil sich Speicherinhalte auch ohne Codeausführung auslesen lassen, sollten Sie das auch ohne konkreten Verdacht erwägen, wenn Ihr System längere Zeit ungepatcht

© 2026 1blu AG <info@1blu.de> |

aus dem Internet erreichbar war. Laut Sansec können dabei sogar TLS-Schlüssel offengelegt werden. Läuft PHP im selben Prozess wie die TLS-Verschlüsselung, etwa bei Apache mit mod_php oder bei FrankenPHP, sollten Sie deshalb auch das Zertifikat mit einem neuen Schlüssel neu ausstellen lassen und das alte widerrufen.

Wo finde ich weitere Informationen?

Verbindliche Angaben kommen vom PHP-Projekt und von Ihrem Distributor. Konkrete Konfigurationsbeispiele entnehmen Sie bitte den folgenden Quellen und prüfen Sie diese vor dem Einsatz in Ihrer Umgebung. Die meisten Quellen sind englischsprachig.

Offizielle Stellen

- PHP-Projekt: Sicherheitshinweis vom 21. Oktober 2026 mit Downloads, Changelogs und Übergangsmaßnahmen; Support-Zeiträume unter Supported Versions
- Technische Sicherheitshinweise des PHP-Projekts auf GitHub: CVE-2026-109618 und CVE-2026-109619
- CERT-EU: Security Advisory 2026-018
- BSI: „PHP: Kritische Schwachstellen bedrohen Webanwendungen weltweit“ (Kurzfassung) und ausführliche Fassung als PDF mit Bewertung und Übergangsmaßnahmen
- NCSC-NL: Liste betroffener Produkte, die PHP enthalten
- Sicherheitsmeldungen Ihrer Linux-Distribution bzw. Ihres Hosting-Panels, z. B. Ubuntu, Debian und Red Hat

Technische Berichte der Entdecker

- Calif: A tale of two PHP remote vulnerabilities
- Wiz Research: PHP2Shell: AI uncovers critical RCE vulnerabilities affecting 70% of the Internet (mit dem Angriffsweg über den OPcache)

Hintergrund und Mitigationen

- Mattias Geniar: PHP CVE-2026-109618 and CVE-2026-109619: early mitigation steps (Stand der Fakten, umstrittene Punkte, Konfigurationsbeispiele) und … how to apply the patches (Paketstatus der Distributionen, Neustarts, Container)
- Tenable: php2shell FAQ
- Huntress: Rapid Response: php2shell (mit Hinweisen zur Erkennung)
- Sansec: PHP2Shell: Critical PHP RCE (CVE-2026-109618, CVE-2026-109619)
- Kevin Beaumont: Beitrag zu AppArmor und SELinux als Schutzschicht (Mastodon)
- The Hacker News: PHP Issues Emergency Patches for Two Critical PHP2Shell RCE Flaws as Exploit Attempts Emerge
- SecurityWeek: PHP2Shell: Mass Exploitation Looms for Leaked Critical PHP Flaws
- heise online: php2shell: Kritische Sicherheitslücken erlauben Codeeinschleusung in PHP

Suchbegriffe für die eigene Recherche

- „CVE-2026-109618“ bzw. „CVE-2026-109619“ zusammen mit dem Namen Ihrer Distribution oder Ihres Panels
- „php_admin_value memory_limit php-fpm pool“
- „session.upload_progress.enabled php-fpm pool“
- „AppArmor php-fpm profile deny exec“
- „SELinux httpd php-fpm policy“ bzw. „SELinux httpd booleans“

© 2026 1blu AG <info@1blu.de> |

Eindeutige ID: #1892
Verfasser: n/a
Letzte Änderung: 2026-10-22 08:05

© 2026 1blu AG <info@1blu.de> |