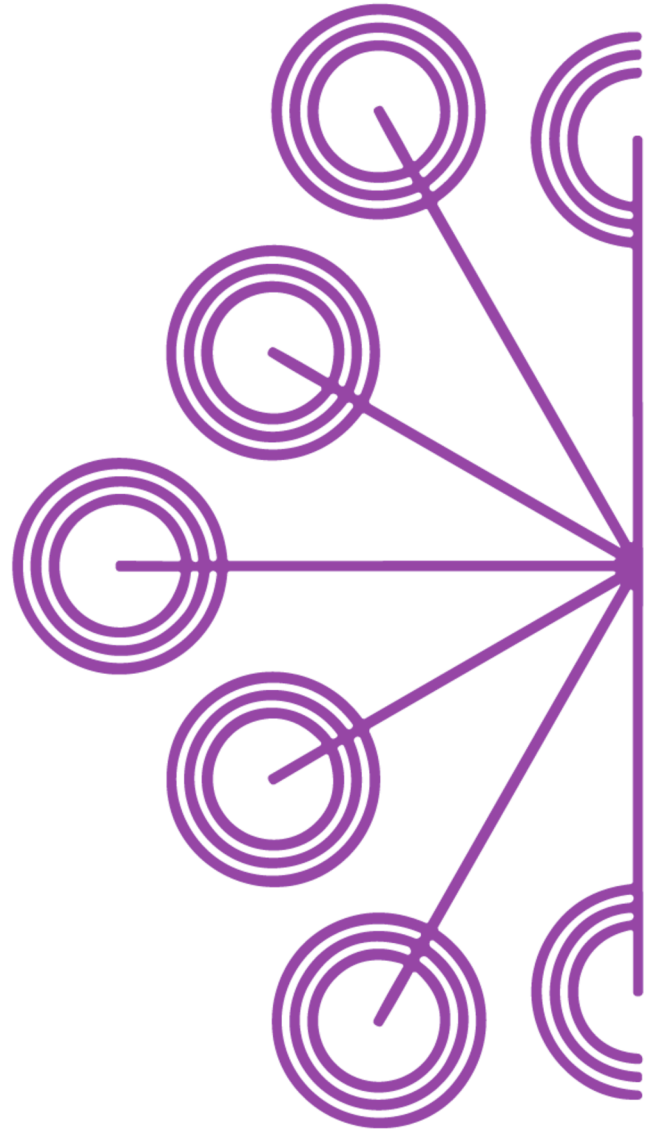




INFORME TÉCNICO · VULNERABILIDAD CRÍTICA

PHP2Shell

Dos fallos de ejecución remota de código (RCE) en el motor de PHP (CVE-2026-109618 y CVE-2026-109619): cómo funcionan, qué demuestran las PoC y cómo defenderse.

CONTENIDO

01	Resumen ejecutivo	03
02	Anatomía de la vulnerabilidad	04
02.1	Las dos vías de ejecución	05
02.2	Por qué no requiere autenticación	06
03	Alcance e impacto	06
04	PoCs, explotación y el factor IA	07
05	Mitigación y respuesta	08
05.1	Checklist de respuesta	09
06	El borde: CDN/WAF	09
06.1	¿Es obligación del WAF?	10
07	Lecciones y fuentes	11

01

Resumen ejecutivo

el problema está en el motor de PHP.

PHP2Shell agrupa dos vulnerabilidades críticas en el *runtime* de PHP (CVE-2026-109618 y CVE-2026-109619), ambas con una puntuación CVSS v4.0 de 9,2. Una sola petición HTTP, sin autenticación, puede bastar para ejecutar código en el servidor. Como el problema reside en el propio intérprete, una aplicación sin fallos en su código sigue estando expuesta.

9,2

CVSS v4.0
ambas, crítica

2

CVE encadenadas
RCE sin login

~70%

de la web
usa PHP

5.4

versión mínima
afectada

HALLAZGO CLAVE

Hay PoC pública y la ventana entre divulgación y ataque se mide en horas. Parchea a 8.5.12 / 8.4.27 / 8.3.36 / 8.2.35 y reinicia los *workers*.

02

02 Anatomía de la vulnerabilidad

No hace falta conocer los detalles de bajo nivel para entender el problema. Ambas CVE comparten una misma raíz: cómo reacciona PHP cuando se queda sin memoria a mitad de una petición.

- ⦿ Las dos vías de ejecución 05
- ⦿ Por qué no requiere autenticación 06

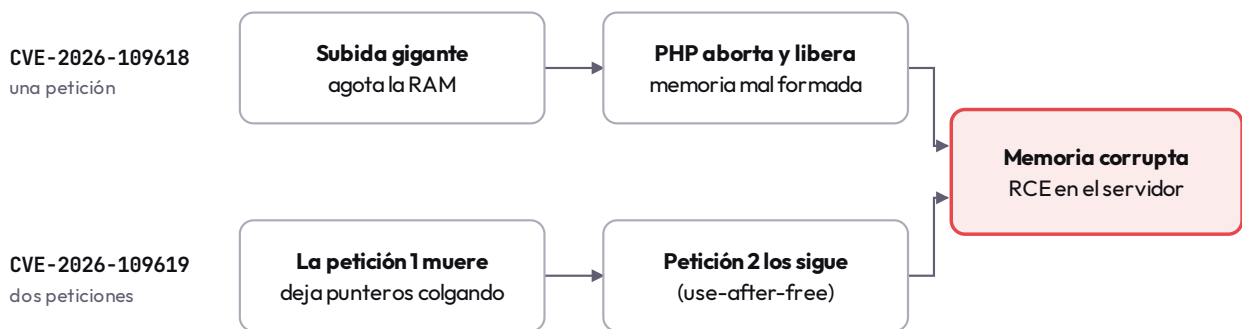
02.1 Las dos vías de ejecución

Cuando una petición agota la memoria, PHP no ordena su espacio de trabajo: abandona la tarea a medias, salta a un punto de recuperación y se salta la limpieza. Quedan referencias que apuntan a memoria que ya no existe. Las dos CVE aprovechan ese hueco por caminos distintos.

FIG-01 Cómo cada vía llega a la ejecución de código

Dos caminos distintos hacia la ejecución de código

Ambas vías explotan que PHP abandona la limpieza tras un error fatal



Fuente: Bitnaga · Análisis sobre PHP.org y Harness

CVE-2026-109618 · una petición

Una subida sobredimensionada agota la RAM antes de que PHP termine de preparar la petición. Al abortar, su limpieza libera memoria a medio construir y la corrompe. En Apache con mod_php ocurre durante el arranque, antes de cargar el código de la aplicación.

CVE-2026-109619 · dos peticiones

Un *worker* atiende muchas peticiones por turnos y guarda punteros globales — como el de `session.upload_progress`, activo por defecto. Una petición que muere a medias los deja apuntando a memoria ya liberada; la siguiente los sigue. Es un *use-after-free* (CWE-416) y funciona también en PHP-FPM.

02.2 Por qué no requiere autenticación

En el montaje más común (Apache con mod_php), PHP procesa la subida maliciosa durante su propio arranque, antes de ejecutar una sola línea del código de tu sitio. Login, permisos y validaciones no llegan a intervenir: el ataque se resuelve por debajo de la aplicación.

03 Alcance e impacto

El código vulnerable se remonta a PHP 5.4, así que casi toda versión en uso está afectada. PHP 8.1 y anteriores ya no reciben parche oficial. No todas las configuraciones corren el mismo peligro:

TBL-01 Exposición por tipo de servidor

TIPO DE SERVIDOR (SAPI)	RIESGO	DETALLE
Apache · mod_php	 CRÍTICO	Ambas vías funcionan y el ataque es más fácil. El montaje clásico de miles de CMS.
PHP-FPM	 ALTO	109619 funciona; 109618 cuaja menos porque el <i>worker</i> suele morir antes.
php-cgi · FrankenPHP · LiteSpeed	 EN ALCANCE	Explotación más difícil o menos probada, pero PHP los declara en riesgo.

Fuente: PHP.org · Canadian Centre for Cyber Security · 24.10.2026

Si el ataque tiene éxito, el atacante ejecuta comandos con los permisos del servidor web: credenciales de base de datos, claves de API, puertas traseras y movimiento lateral. Y apenas deja rastro — en los logs sólo aparece un error rutinario de memoria agotada.

04

PoCs, explotación y el factor IA

la IA acortó de meses a días el salto del bug al exploit.

Salió a la luz una herramienta de prueba de concepto llamada PHP2Shell; watchTowr recreó las vulnerabilidades y Wiz publicó un análisis técnico en profundidad. Alrededor del parche, watchTowr, Patchstack, Cloudflare y Aikido reportaron intentos de explotación, algunos apoyados en una PoC pública.

04.1 Cómo encaja la IA en un pentest

El trabajo ofensivo sigue un bucle: reconocer la superficie de ataque, encontrar un indicio prometedor y — la parte lenta — convertirlo en un exploit fiable, probando variante tras variante y aprendiendo de cada fallo hasta dar con la que funciona. Ese último paso es el que más tiempo y pericia experta exige, y es justo el que la IA automatiza mejor: itera sin descanso sobre miles de hipótesis bajo dirección humana y comprime en días lo que antes llevaba meses.

Es lo que ocurrió aquí: según Wiz, Calif y OpenAI, modelos de IA ayudaron a transformar un bug de memoria sutil en exploits funcionales en días. Cuando el coste de armar un exploit se desploma, lo “demasiado difícil de explotar en la práctica” se convierte en un objetivo viable.

Estado de explotación (en evolución). Al cierre de este informe, PHP y CISA aún no confirmaban explotación exitosa a escala ni figuraban las CVE en el catálogo KEV. Verifica el estado antes de distribuir.

05

Mitigación y respuesta

Parchear es la única solución completa. Actualiza a la versión de tu rama y reinicia el servidor y los *workers*: una app que sigue en pie no significa que el *runtime* se haya recargado.

TBL-02 Versiones corregidas por rama

RAMA	VERSIÓN CORREGIDA	NOTA
8.5	8.5.12	Rama estable actual.
8.4	8.4.27	
8.3	8.3.36	
8.2	8.2.35	Última con soporte de seguridad.
8.1 y anteriores	Sin <i>release</i> oficial	<i>Backport</i> de la distro o migración.

Fuente: PHP.org · 21.10.2026

Parchear no siempre es trivial.

Cambiar de versión puede alterar el comportamiento del código que corre encima. Antes de desplegar, entiende cómo reaccionará tu aplicación — funciones retiradas, cambios entre versiones, dependencias que esperan una versión concreta — y pruébalo fuera de producción. A veces el parche no es opción: software heredado que depende de la retrocompatibilidad puede romperse al subir de rama; ahí el reto es gestionar el riesgo mientras se busca una vía viable.

Mitigaciones temporales (no sustituyen al parche).

- Baja los límites de subida de archivos y de tamaño de POST.
- Desactiva `session.upload_progress.enabled`: sólo frena una vía de 109619 y nada contra 109618.
- Pon una regla WAF delante del tráfico del exploit; trátala como un puente, saldrán variantes.
- Inventaria dónde corre PHP antes de que lo haga un atacante.

05.1 Checklist de respuesta

- 01 Inventaria dónde corre PHP: servidores, contenedores y productos de terceros con PHP embebido.
- 02 Prioriza lo expuesto a internet; dentro de eso, Apache con mod_php.
- 03 Aplica el parche (o el *backport*) y reinicia servidor y *workers*.
- 04 Identifica instalaciones *end-of-life* (8.1 o anterior) y planifica la migración.
- 05 Si no puedes parchear ya, aplica las mitigaciones temporales.
- 06 Monitoriza procesos de PHP que lancen *shells*, intérpretes o descargas.
- 07 Verifica la versión efectiva tras el reinicio, no sólo la instalada.

06 El borde: CDN/WAF

el WAF gana tiempo mientras despliegas el parche.

El borde es el sitio idóneo para una mitigación: una sola regla protege miles de orígenes en minutos, y los grandes proveedores desplegaron firmas incluso antes de la divulgación. Pero un WAF perimetral sólo ve el tráfico que pasa por él y lee el *multipart* de forma distinta a como lo hace PHP: es una capa extra, no una cura.

TBL-03 Respuesta de los proveedores *edge*

PROVEEDOR	MEDIDA DESPLEGADA	ALCANCE Y LÍMITE
Cloudflare	Reglas gestionadas activas por defecto (incluido Free) desde el 21.10; inspección de petición y respuesta, con 403 antes de llegar al atacante.	Sólo tráfico <i>proxied</i> ; descrito como paliativo.
AWS WAF	SizeRestrictions_BODY (bloquea cuerpos > 8 KB) para 109618; regla personalizada "PHP2Shell" para 109619.	Controles compensatorios; actualizan su <i>managed rule group</i> .
Vercel	Reglas WAF desplegadas antes de la divulgación para patrones conocidos del exploit.	No cubre peticiones directas a un <i>backend</i> externo.

Fuente: avisos de Cloudflare, AWS y Vercel · 21.10.2026

06.1 ¿Es obligación del WAF parar el acceso?

No hay una obligación universal de que el WAF *garantice* el bloqueo: rige la responsabilidad compartida. El proveedor protege el perímetro con diligencia; remediar el *runtime* es de quien opera el sistema. Marcos como PCI-DSS, NIS2 o el art. 32 del RGPD exigen "medidas apropiadas", y un WAF ayuda a cumplirlas — pero junto al parche, no en su lugar. Como el *bypass* es plausible (un WAF lee el *multipart* distinto a PHP) y php2shell es una propiedad del motor con variantes por venir, confiar el cierre sólo al WAF sería lo contrario a una medida apropiada.

HALLAZGO CLAVE

Ningún WAF "cierra" php2shell: reduce exposición y compra horas. El cierre llega con el parche y el reinicio — y bloqueando además el acceso directo al origen para que nadie esquive el borde.

Lecciones y fuentes

Dependencia silenciosa del *open source*. Una pieza invisible sostiene buena parte de internet; cuando falla, falla todo a la vez. Es el patrón de Log4Shell y Heartbleed, y se repetirá.

El reto es organizativo, no técnico. Parchear es fácil cuando el código lo permite; lo difícil es encontrar cada instalación y saber quién la mantiene. PHP2Shell es una prueba de si sabes qué software corre en tu entorno.

La IA cambia el cálculo de riesgo. Si armar un exploit cuesta días y no meses, ya no basta con apostar a que nadie se tomará la molestia.

php2shell es infraestructura compartida con la barrera de explotación rebajada.

Lo urgente es mecánico — parchea, reinicia, inventaria. Lo importante a medio plazo es asumir que la distancia entre “se ha descubierto un bug” y “nos están atacando” se mide ya en horas.

FUENTES

Harness — php2shell: the flaw hiding all across the web — harness.io/blog/php2shell-the-flaw-hiding-all-across-the-web

Cloudflare / cobertura del despliegue WAF php2shell — [blog.cloudflare.com](https://blog.cloudflare.com/php2shell/) · securityweek.com

AWS — Security Bulletin 2026-143-AWS — aws.amazon.com/security/security-bulletins/2026-143-aws

Vercel — php2shell security bulletin — vercel.com/kb/bulletin/php2shell-security-bulletin

Dark Reading — 'PHP2Shell' Bugs Signal Red Alert — darkreading.com/application-security/php2shell-bugs-red-alert-security-industry

Sonatype — PHP2Shell Makes PHP Inventory and Patch Ownership Urgent — sonatype.com/blog/php2shell-makes-php-inventory-and-patch-ownership-urgent

Canadian Centre for Cyber Security — Alerta AL26-026 — cyber.gc.ca/en/alerts-advisories/al26-026

NVD — CVE-2026-109618 / CVE-2026-109619 — nvd.nist.gov/vuln/detail/cve-2026-109618

Datos verificados a 24.10.2026. Por tratarse de un incidente en curso, conviene contrastar el estado de explotación y los avisos oficiales antes de su publicación.