contact@namiq.io · namiq.io

NamiQ

SECURITY ADVISORY

PHP2Shell

CVE-2026-109618 and CVE-2026-109619 · PHP

Assessment & Action Plan

Two critical bugs in PHP itself affect almost every PHP release since 2012. Which versions fix them, how to check a server without being fooled by your package manager, and what to do if you cannot patch today.

PREPARED BY	Hung Le, NamiQ	DATE	2 November 2026
FACTS AS OF	22 Oct 2026; CISA 26, 28 Oct	VERSION	1.4
SEVERITY	**Critical: CVSS 4.0 9.2; both in CISA KEV**	STATUS	Fixed in PHP 8.5.12, 8.4.27, 8.3.36, 8.2.35

NamiQ © 2026 NamiQ. All rights reserved. This document is the property of NamiQ. It may be shared unaltered, with attribution.
Companion article: namiq.io/insights/php2shell-cve-2026-109618-109619

Update, 2 November 2026: the note now ends with Step 5, which checks every server and project with lattice-scan, and the analysis report is reissued as version 1.4 with the CISA listing and the same step. It also gains a section on how these flaws chain with Fragnesia, with the attack chain stage by stage, and a table and picture for IT admins, AI agent skills and CI/CD. The facts above are otherwise as of the dates given.

Update, 28 October 2026: CISA has also added Linux kernel flaw CVE-2026-46300 (Fragnesia) to the catalog, with a due date of 31 October, and says in the entry that it can be chained with these two PHP CVEs to implant PHP webshells and run commands as root. If you run PHP on Linux, patch the kernel as well as PHP. We cover the kernel flaw, and what the chain changes, in a separate note.

Update, 26 October 2026: CISA has added both CVEs to its Known Exploited Vulnerabilities catalog, with a due date of 29 October. The note below was written on 22 October, when neither was listed; where it says so, read it with this update. The advice does not change: patch, or take internet-facing PHP offline until you can. Source: CISA KEV catalog.

On 21 October 2026 the PHP project shipped an out-of-band security release for two critical flaws in the PHP engine itself: CVE-2026-109618 and CVE-2026-109619, together nicknamed PHP2Shell. They are not bugs in WordPress, Laravel or your own code. They are in the runtime underneath all of them, so a clean application on an unpatched PHP is still exposed.

This note is for you if you run, host or are responsible for anything written in PHP and need to decide what to do this week. Everything below is as of 22 October 2026, and every claim links to its source at the end. The story is moving fast. Check the sources before acting on anything time-sensitive.

The short version

- Fixed releases: PHP 8.5.12, 8.4.27, 8.3.36 and 8.2.35. Anything earlier on those branches is vulnerable.
- Affected range: the PHP project and Google put it at 5.4 through 8.5.11, so roughly every PHP release since 2012.
- PHP 8.1 and older get no new official release. Your fix there comes from your Linux distribution or an LTS vendor, or from upgrading to 8.2 or later.
- Severity: 9.2 (Critical, CVSS 4.0) from the PHP security team, with no login and no user interaction needed.
- Exploitation: attempts were seen before and on release day. Update, 26 October: CISA added both CVEs to its Known Exploited Vulnerabilities catalog, with a remediation due date of 29 October 2026. Ransomware use is not recorded. Working attacks against a default install had not been shown publicly when we last checked.
- Patching is the only complete fix. The workarounds reduce exposure; they do not close it.
- On Linux, patch the kernel as well. CISA says the kernel flaw CVE-2026-46300 (Fragnesia) can be chained with these two to plant a webshell and run commands as root. The chain is below.

What the two bugs actually are

CVE-2026-109618 ("RCE via OOM corruption"). When PHP runs out of memory, it aborts the request by jumping straight out of the code that was running. If that happens while internal request data is only partly set up, the cleanup that follows can read or act on data that was never valid. The PHP advisory says this can lead to remote code execution. Malformed multipart uploads are one way to push a request into that state: a missing content length, an unlimited number of part headers, or empty file fields.

CVE-2026-109619 ("RCE via stale global state"). PHP can keep internal state, such as error, session or extension state, pointing at memory that belonged to a request that has already finished. When the same worker process picks up the next request, some code runs before that state is reset and can read the freed memory, a classic use-after-free. The advisory names two confirmed paths. One is PHP 8.5's fatal-error backtraces. The other is session upload-progress handling. For this one to work, the attacker's two requests have to land on the same worker.

In plain terms, the first bug can corrupt memory within one request, and the second can carry corruption from one request into the next. Researchers chained them to get code execution.

Who is exposed

The PHP project's position is that every installation serving web requests is at risk: PHP-FPM, Apache with mod_php, php-cgi, FrankenPHP and LiteSpeed. Risk is highest with mod_php, where a demonstrated exploit fires from an unauthenticated request before any of your application code runs.

Researchers have confirmed pre-authentication code execution against specific versions of WordPress, Drupal, Joomla and Magento. That is not the full list. Those platforms are named because they are the ones people tested first.

Places PHP hides that teams forget to count:

- Container images you built yourself. Rebuilding on the same base image tag may not pull a patched PHP.
- Managed platforms that do not patch automatically. On Google Cloud, App Engine flexible must be redeployed, and Cloud Run is patched only if automatic base-image updates are switched on. Other clouds have similar splits.
- Hosting control panels, admin tools and vendor appliances with PHP inside. Check each vendor's advisory. Fortinet and F5 have already published theirs.
- Old internal tools on 7.x that nobody owns, which are exactly the ones with no upstream fix.

How bad is it, really?

This is where the early coverage disagreed, so here is each position and where it came from.

Severity. The PHP security team rates both bugs 9.2 under CVSS 4.0. Its vector marks attack complexity as high and says attack prerequisites are present. That is critical, but not trivial. CVE IDs that circulated before release carried a 10.0, which the PHP Foundation says was incorrect. Some vendors quote 9.8 under the older CVSS 3.1. Several early summaries, including the first internal draft of this note, called exploitation low-complexity. The official vector says otherwise.

Exploitation. Amazon Threat Intelligence saw an attempt against its honeypots on 17 October, four days before disclosure, and further attempts from what looked like multiple actors on 21 October. Cloudflare reported early attacks on the morning of 21 October, hours before the 14:00 UTC release. watchTowr reports exploitation attempts against both bugs. On the other side, CISA's alert said it had not observed exploitation when it published. Neither CVE was in CISA's Known Exploited Vulnerabilities catalog (version 2026.10.21) when we checked it on 22 October.

Working exploits. The PHP security team found no functional exploits at release. On 22 October Sansec revised its view to say mass exploitation now looks less likely: the public proof-of-concept code it examined does not work against a default PHP install, and most of it needs a separate file-inclusion bug in the target application. Some exploit code circulating has not been verified and may contain malware. Do not run it to test yourself.

> *Attempts are real; reliable mass exploitation has not been shown. That is a reason to patch on a plan, starting today with whatever faces the internet. It is not a reason to wait.*

One recommendation deserves weight because it changes what you look for. Amazon advises starting any investigation from 17 October, not from the disclosure date. If an internet-facing PHP server stayed unpatched over that period, look at its logs from then onwards.

How these flaws chain with the Linux kernel flaw Fragnesia

On 28 October CISA added the Linux kernel flaw CVE-2026-46300, nicknamed Fragnesia, to its Known Exploited Vulnerabilities catalog (due 31 October) and said in the entry that it can be chained with these two PHP flaws to implant PHP webshells and run commands as root. The reason it fits is simple. A PHP flaw gives an attacker the web server's account on the machine. Fragnesia is a local privilege escalation, scored 7.8 under CVSS 3.1 for needing only local access and low privileges, and the web server's account is exactly that foothold. Alone, each is a smaller problem: PHP2Shell

without it leaves the attacker as an unprivileged user, and Fragnesia without it needs a way onto the machine. Together they take an attacker from an internet request to root.

```
Internet request
   |  1. PHP2Shell: CVE-2026-109618 / CVE-2026-109619 (PHP, no login)
   v
Code runs as the web server's account
   |  2. A PHP webshell is written to a web-writable folder (CISA)
   v
Commands run as the web account, and stay available
   |  3. Fragnesia: CVE-2026-46300 (Linux kernel, local, low privilege)
   v
Commands run as root (CISA)
```

The chain, stage by stage. Stage 1 and 2 are from the PHP advisories and CISA; stage 3 is from the CVE record and a third-party write-up.

Stage	What happens	What breaks it	What to look for
1. Entry: PHP2Shell	An unauthenticated multipart request makes PHP corrupt memory (CVE-2026-109618) or reuse stale state (CVE-2026-109619), and researchers chained them to code execution as the web account.	Patch PHP to 8.5.12, 8.4.27, 8.3.36 or 8.2.35 and restart every worker; until then block multipart requests or take the system offline.	Multipart requests with odd structure, bursts of out-of-memory fatal errors, worker crashes.
2. Foothold: webshell	CISA says PHP webshells are implanted. A webshell is a script left in a web-writable folder that runs commands on request, so access survives after the first request.	Patching PHP does not remove a shell already planted. Make web directories read-only to the web account where the application allows it, and rebuild a host that has one.	New or changed PHP files in web roots and upload directories; PHP workers starting shells or fetching tools.
3. Escalation: Fragnesia	The kernel merges network buffers and loses a marker that says the memory belongs to someone else. Decrypting ESP traffic then writes over the file cache, so a cached copy of a setuid program can change while the file on disk does not. The next run of that program gives root. The write-up says the proof of concept needs the espintcp module.	Update the kernel and reboot, or use a vendor live patch. Stop-gap: block esp4, esp6 and rxrpc, which breaks IPsec on hosts that use it.	Root activity that no administrator owns: new accounts, SSH keys, scheduled jobs, services. A disk integrity check will not see an altered cache.

4. Result: root	Commands run as root. On a container host, the write-up argues that a compromised container can reach the host.	Nothing at this stage prevents it; this is why stages 1 to 3 matter.	Rotate every secret the host could read and rebuild from a known-good image rather than clean in place.

What the chain changes. On a Linux host that runs PHP, patching PHP is no longer enough and patching the kernel is no longer a later job: the two fixes close two ends of one path, and the kernel one needs a reboot or a live patch. Rank hosts that run PHP first for both. If a host was exposed to the PHP flaws unpatched, treat it as possibly compromised to root and look for evidence the chain ran, because the second step now has a CISA listing of its own.

What we know and what we do not. The chain is CISA's statement in its catalog entry. We found no independent write-up of how the steps join, so laying them end to end is our reading of the entries and the write-up, and we have not run any of it. Whether it has been used in the wild is not stated.

Step 1 — Find out which version you are actually running

This is the step most likely to give you a false answer. Debian, Ubuntu, Red Hat and others backport security fixes into their existing packages without changing the version number. php -v can report 8.1.2 on a fully patched Ubuntu server and 8.3.20 on an unpatched one. Read the package changelog, not the version string:

```
# Version as reported (useful, but not proof either way on distro packages)
php -v

# Debian / Ubuntu: is the fix in the installed package?
zgrep -l CVE-2026-109618 /usr/share/doc/php*-common/changelog.Debian.gz

# RHEL / Alma / Rocky / Fedora
rpm -q --changelog php-common | grep -E 'CVE-2026-10961(8|9)'

# Containers: check the image that is running, not the Dockerfile
docker exec <container> php -v
```

Check what the web server uses, not what the command line uses. PHP-FPM and the CLI can be different binaries with different configuration. Confirm the version and settings from a request served through FPM, for example a phpinfo() page on localhost that you delete straight afterwards.

Step 2 — Patch, then restart every worker

- On 8.2 to 8.5: upgrade to 8.2.35, 8.3.36, 8.4.27 or 8.5.12, or install your distribution's patched package.
- On 7.4 to 8.1: use your distribution's or LTS vendor's backported package if you have one. The PHP security team has published minimal source backports for these branches but will not cut releases. Plan the move to 8.2 or later regardless.

- On 5.4 to 7.3: there is no fix from the PHP project. Upgrade, isolate, or pay for an extended-support vendor.
- Restart PHP-FPM, Apache and any long-running PHP worker. A patched binary on disk does nothing for a process that loaded the old one.
- Rebuild container images from a patched base and redeploy. Then check the running image, not the build log.
- On Linux, update the kernel and reboot (or use a vendor live patch) for CVE-2026-46300, starting with the hosts that run PHP. See the chain above.

Step 3 — If you cannot patch today

CISA's advice is blunt: if you cannot patch right away, or you run PHP 8.1 or earlier, take the PHP system offline until it is updated. That is the safe default for anything internet-facing. Sansec, looking at the exploit code available on 22 October, judges that most online stores do not need to go offline. Which one you follow is a risk decision for you to make. What matters is that someone makes it deliberately rather than by default.

If a system has to stay up, both PHP advisories are explicit that these workarounds reduce exposure but do not make an installation safe. Use them to buy days, not weeks, and restart every worker after any configuration change:

- Block multipart/form-data requests at the web server or proxy if the application does not need file uploads. Every known trigger uses multipart. The cost is that uploads break, so test any upload, cart or checkout flow, and remove the block after patching.
- Set session.upload_progress.enabled = 0. This blocks the demonstrated session path of the second bug. PHP suggests keeping it off even after patching if you do not use the feature.
- On PHP 8.5, set fatal_error_backtraces = 0.
- Keep post_max_size conservative, cap request body size at the proxy, and lower max_input_nesting_level and the limits on input variables, multipart parts and uploads.
- PHP-FPM: consider pm.max_requests = 1 so each worker handles one request and stale state cannot carry into the next. It costs throughput, so measure it.
- memory_limit = -1, enforced so applications cannot override it, removes the out-of-memory trigger for the first bug. It also removes PHP's guard against a runaway script, so one request can exhaust the server. Watch memory closely and restore the limit after patching.
- Do not use RLIMIT_AS or RLIMIT_DATA as a memory cap. The PHP advisory warns they trigger the vulnerable path.

A web application firewall helps at the edge, but only for traffic that passes through it. Cloudflare has deployed managed rules, available on its free plan too, and calls them a stopgap. Google gives equivalent Cloud Armor guidance. Neither protects an origin that can be reached directly.

Step 4 — Check whether it already happened

- Before you rebuild a server you suspect, preserve its logs and, if you can, a disk image. Patching and redeploying destroy the evidence you would need later.
- From 17 October onwards, look for multipart requests with unusual structure: huge or missing content lengths, very many parts or headers per part, empty file fields. Look also for bursts of out-of-memory fatal errors in the PHP log, and for worker crashes or segmentation faults.
- On the host, look for PHP worker processes starting shells or downloading tools (CISA calls this out specifically), outbound connections from web workers, and new files in web-writable directories.
- Assume the chain may have run on an unpatched PHP host: look for new PHP files in web roots, then for root activity that no administrator owns (new accounts, SSH keys, scheduled jobs, services). A disk integrity check will not show the kernel step.
- Your scanner and EDR vendors are shipping detections: Tenable, Google Security Operations, Wiz and others have published theirs. Make sure they are switched on, then treat a clean scan as one input, not proof.

Step 5 — Check every server and every project, and keep the evidence

Steps 1 to 4 are done by hand, one server at a time. To do the same over a fleet, and to keep a record, use the open-source lattice-scan command line. Run on the server itself, it reads the installed packages, snaps, the names of the running container images and the kernel, compares them with the CVE ids you name, and sends only those ids out. Naming the CVEs needs no key.

```
# on each PHP server, after patching and restarting
lattice-scan --cve CVE-2026-109618 --cve CVE-2026-109619
```

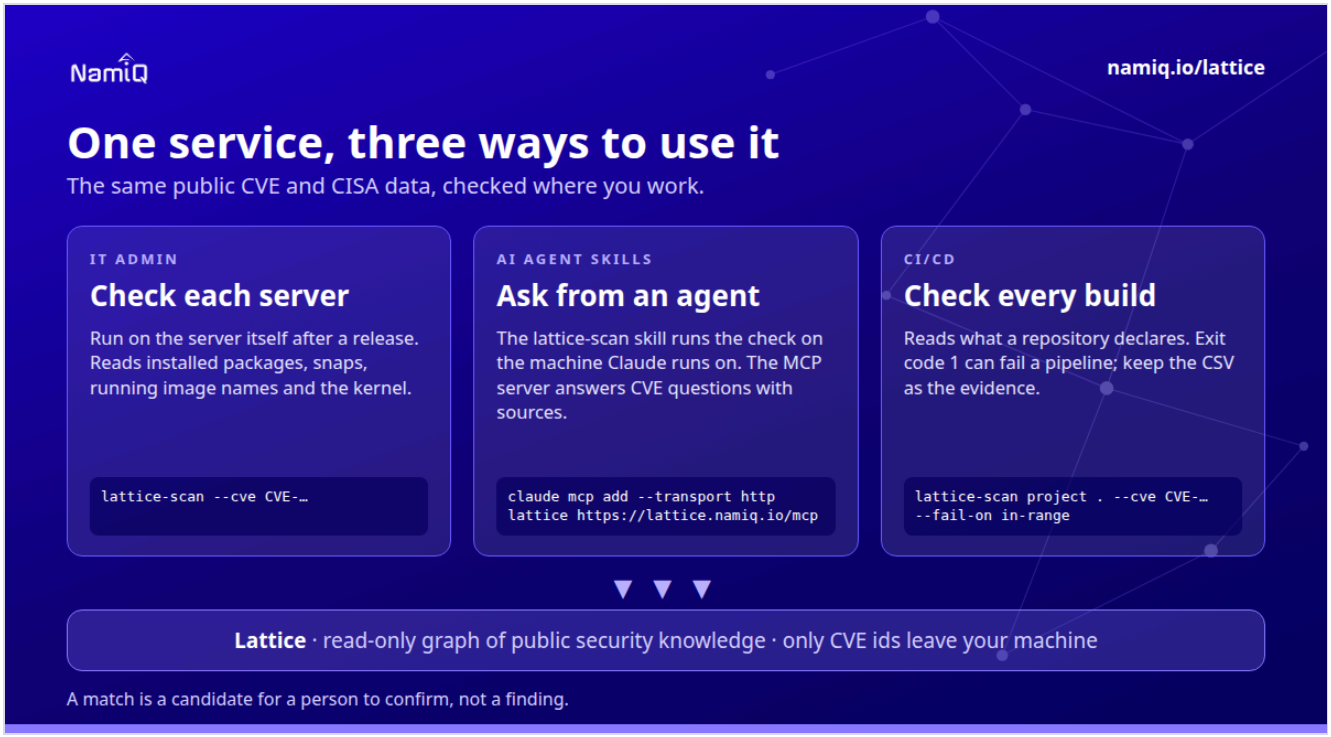
A match is a candidate for a person to confirm, not a finding. A distribution can fix a CVE under an unchanged version number, which is the same trap as in Step 1, so on Debian-family hosts read the changelog line the report shows. It reads the names of running images, not what is inside them. Run it from the server, not from the laptop or CI runner that deploys to it, and keep its output with the change record.

For the code you ship, the project check reads what a repository declares. It reads Dockerfiles and compose files, such as a FROM php:8.3 line or an apt-get install of PHP packages, and it does not read composer.json, so a PHP application's Composer dependencies are not covered; the base image is. A tag without a patch number cannot show whether the fix is in, so confirm the image that is actually running.

```
lattice-scan project . --cve CVE-2026-109618,CVE-2026-109619
```

Who can use it, and how

For	What to do	Command
IT admin	Run the host check on each server after a release, and again when a new entry lands.	lattice-scan --cve CVE-2026-109618 --cve CVE-2026-109619
AI agent skills	The lattice-scan skill runs the check on the machine Claude runs on. The Lattice MCP server answers questions about the CVEs with their sources.	claude mcp add --transport http lattice https://lattice.namiq.io/mcp
CI/CD	Check the repository on every pull request. Exit code 1 fails the pipeline; keep the CSV as the evidence.	lattice-scan project . --cve CVE-2026-109618,CVE-2026-109619 --fail-on in-range



Where the same check is used. The three ways read the same public data; only CVE ids leave your machine.

Guide: namiq.io/lattice/guides/choose-your-channel. MCP server (io.namiq/lattice in the MCP registry): namiq.io/lattice. Package: pypi.org/project/lattice-scan, or namiq.io/apt.

Reference: should you patch now?

Yes. Treat internet-facing PHP as a P0 change. The decision for each system:

Your situation	Recommended action	Target
Internet-facing, supported branch (8.2 to 8.5)	Upgrade to the fixed release, restart all workers, verify through the web server (Step 1).	Within 24 hours; mod_php first

Internet-facing, 7.4 to 8.1 on a distribution or LTS package	Install the vendor's backported package, restart, and verify through the changelog, not php -v.	Within 24 hours
Internet-facing, 8.1 or older with no vendor fix	CISA: take it offline until updated. Otherwise isolate it behind strict controls and start the upgrade.	Today
Cannot patch yet (change freeze, testing)	Make a deliberate, owned risk decision: offline, per CISA, or compensating controls. Record who decided and when it expires.	Today; days, not weeks
Internal-only systems	Patch in the next change window.	Within 72 hours
Vendor appliances	Follow each vendor's advisory; do not patch PHP inside an appliance yourself.	As vendors publish

Targets are NamiQ's recommendation for a typical small or mid-size estate. CISA's listing of 26 October set a 29 October deadline for US federal agencies; it is not a deadline for anyone else.

Reference: action plan by urgency

Immediate (0 to 4 hours)

- List every internet-facing PHP service, with its SAPI and the version verified in Step 1.
- Patch them, mod_php first, and restart every PHP worker. A patched binary on disk does nothing for a process that loaded the old one.
- For any you cannot patch, make the offline-or-compensate decision explicitly and give it a named owner.
- Switch on edge protection where you have it: Cloudflare's managed rules (available on the free plan; Cloudflare calls them a stopgap) or Google Cloud Armor's recommended rules.
- On any system you suspect, preserve logs and, if possible, a disk image before patching. Rebuilding destroys evidence.

Short term (4 to 24 hours)

- Patch internal PHP: intranet apps, CI/CD, monitoring and admin tools.
- Rebuild container images from a patched base, redeploy, and verify the running image.
- Redeploy managed-platform services that are not patched automatically.
- Collect vendor advisories for every appliance or product with embedded PHP.
- Keep session.upload_progress.enabled=0 wherever the feature is unused; PHP recommends this even after patching.
- Run the host check (Step 5) on each patched server and keep the output.

Medium term (24 to 72 hours)

- Hunt for compromise from 17 October onwards (table below).
- Scan to confirm: Tenable Nessus plugin 369483, Web App Scanning plugins 115719 to 115722, and equivalent checks in other scanners. A clean scan is one input, not proof.
- Schedule upgrades off 8.1 and older; 8.2 itself reaches end of life on 31 December 2026.
- Record both CVEs in your vulnerability register, with an owner and evidence of closure for every system.

Ongoing

- Both CVEs are in CISA's catalog: re-run the hunt whenever a new advisory or indicator appears, including the Linux kernel chain in the 28 October update.
- Add a pipeline check that fails builds whose PHP is below the fixed baseline (the project check of Step 5 covers Dockerfiles).
- Set a patch SLA for critical, unauthenticated RCE in shared runtimes. Under 24 hours for internet-facing systems is a reasonable target.

Reference: patch procedure per host

- Take a backup or snapshot; on suspect hosts, preserve evidence first.
- Install the fixed release or the vendor's backported package.
- Restart PHP-FPM, the web server and any PHP workers; confirm that no old worker processes remain.
- Verify through the web server and the package changelog.
- Remove temporary workarounds that cost functionality (for example a multipart block), but keep upload progress off if unused.
- Record the verification output as evidence of closure.

Reference: compensating controls if you cannot patch today

Both PHP advisories are explicit: these reduce exposure but do not make an installation safe. Restart all workers after any change, and treat them as days of cover, not weeks.

Control	Addresses	Cost and caveats
Block multipart/form-data at the proxy or web server	All known triggers (both CVEs)	File uploads break. Test checkout, cart and admin uploads. Remove after patching.
session.upload_progress.enabled = 0	109619 session path	Low cost. Disabling it alone is not enough (Sansec).
fatal_error_backtraces = 0 (PHP 8.5)	109619 backtrace path	Low cost. Leaves other stale-state paths open.

memory_limit = -1, enforced so apps cannot override it	109618 OOM trigger	Removes PHP's guard against runaway scripts. One request can exhaust the server, so monitor memory and restore the limit after patching.
Conservative post_max_size; cap decoded body size at the front end	OOM reachability	Can reject legitimate large uploads.
Lower max_input_nesting_level; limit input variables, multipart parts, headers per part and uploads	OOM reachability	Needs testing against real forms.
pm.max_requests = 1 (PHP-FPM)	109619 same-worker requirement	Costs throughput; measure before rolling out.
Container or cgroup memory.max	Availability boundary	Not a complete mitigation (PHP).
Do not use RLIMIT_AS or RLIMIT_DATA	None	The PHP advisory warns these trigger the vulnerable path.
WAF managed rules (Cloudflare, Cloud Armor)	Known exploit vectors	A stopgap that covers only traffic routed through the WAF. Lock the origin down to the WAF.
Take the system offline	Everything	CISA's advice if you cannot patch or run 8.1 or older. The business cost is yours to weigh.

Reference: detection and hunting (look back to 17 October)

Where	What to look for
Web and proxy logs	Multipart requests with unusual structure: huge or missing content lengths, very many parts, many headers per part, empty file fields. Bursts of such requests from one source, especially against upload endpoints.
PHP and FPM logs	Bursts of 'Allowed memory size exhausted' fatal errors. Worker crashes, segmentation faults, 'child exited on signal' messages, and maximum-execution-time errors on endpoints that normally never hit them.
Host or EDR	PHP worker processes (php-fpm, apache2, httpd) starting shells or downloading tools, which CISA calls out specifically. Outbound connections from web workers; new or modified files in web-writable directories.

Security tooling	Make sure vendor content is enabled: Google Security Operations (Mandiant Frontline Threats rule pack, Emerging Threat Center GLOBAL.26.015), Wiz Threat Intel Center, Tenable plugins. Do not run unverified public exploit code to test yourself; some circulating PoCs may contain malware.

If you find signs of compromise: isolate the host, preserve evidence, rotate every credential the PHP application could read (database, API keys, cloud tokens in environment variables), and rebuild from a known-good image rather than cleaning in place.

What we do not know yet

- Whether ransomware groups use these flaws. CISA added both CVEs to its Known Exploited Vulnerabilities catalog on 26 October (due 29 October) and records ransomware use as not recorded, which is not the same as no ransomware use.
- Whether a reliable exploit against a default configuration appears. Sansec's revised view could change in either direction.
- Whether the PHP and kernel chain has been used in the wild, and how its steps join in detail. CISA states the chain; we found no independent account.
- Which appliances and hosted products embed a vulnerable PHP. Vendor advisories are still arriving.

If you take one thing from this: the hard part is not the patch. It is knowing every place PHP runs, and which of those places has an owner. If that list takes you more than an afternoon to produce, fix that too, because the next runtime-level bug will ask the same question.

Appendix 1. Severity decoded

CVE-2026-109618 and CVE-2026-109619: score 9.2, `CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N` From the PHP security team's advisories (CVSS 4.0). Attack complexity is High and attack requirements are Present: conditions outside the attacker's control must hold.

Metric	Value	What it means here
Attack vector	Network	Reachable over a network, with no need to be on the same segment.
Attack complexity	High	Needs conditions outside the attacker's control.
Attack requirements	Present	Needs a deployment condition the attacker does not control.
Privileges required	None	No account is needed.
User interaction	None	No one has to do anything.
Vulnerable system: confidentiality	High	Total loss of confidentiality.
Vulnerable system: integrity	High	Total loss of integrity.
Vulnerable system: availability	High	Total loss of availability.
Subsequent systems: confidentiality	None	No data exposed.
Subsequent systems: integrity	None	Nothing can be altered.
Subsequent systems: availability	None	No effect on availability.

CVE-2026-46300 (Fragnesia, the kernel flaw in the chain): score 7.8, `CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H` From the CVE record. Local, low privileges: the score describes the kernel flaw on its own, not the chain. The chain is CISA's statement.

Metric	Value	What it means here
Attack vector	Local	Needs an account or code already running on the machine.
Attack complexity	Low	No special conditions beyond the flaw itself.
Privileges required	Low	An ordinary account is enough.
User interaction	None	No one has to do anything.
Scope	Unchanged	Damage stays inside the vulnerable component.
Confidentiality	High	Total loss of confidentiality.

Integrity	High	Total loss of integrity.
Availability	High	Total loss of availability.

Appendix 2. Timeline

Date	Event
17 October 2026	Amazon Threat Intelligence sees an exploitation attempt against its honeypots, four days before disclosure.
21 October 2026	PHP releases 8.5.12, 8.4.27, 8.3.36 and 8.2.35 and publishes both advisories. CISA publishes 'Critical Vulnerabilities in PHP Require Immediate Patching'; it had not observed exploitation at publication.
22 October 2026	Sansec revises its view: mass exploitation is less likely, because the public proof-of-concept it examined fails against a default install.
26 October 2026	CISA adds both CVEs to the Known Exploited Vulnerabilities catalog, due 29 October.
28 October 2026	CISA adds Linux kernel flaw CVE-2026-46300 (Fragnesia), due 31 October, and says in the entry that it can be chained with these two PHP CVEs.

Appendix 3. Exposure by situation

These priorities are our judgement from the attacker prerequisites in the sources. They are not vendor ratings.

Your situation	Priority	Timeframe
Linux host running PHP, unpatched for PHP2Shell and for the kernel flaw CVE-2026-46300	Critical	Patch PHP and the kernel now; reboot the kernel
Apache with mod_php, internet-facing	Highest	Patch within 24 hours
PHP-FPM, php-cgi, FrankenPHP or LiteSpeed, internet-facing	High	Patch within 24 hours
The same, internal only	Medium	Within days
Vendor appliances and panels with PHP inside	Unknown until the vendor says	Read each vendor advisory
PHP CLI only (cron, queue workers)	Not addressed by sources	Normal patch cycle
Already on a fixed release or a distribution package that names the CVE in its changelog	None	Confirm and record

Appendix 4. Definition of done

- Every place PHP runs is listed with an owner.
- Each one is on 8.2.35, 8.3.36, 8.4.27, 8.5.12 or later, or on a distribution package whose changelog names both CVEs.
- On Linux, the running kernel is fixed for CVE-2026-46300 (checked with uname -r and the distribution's changelog), starting with the hosts that run PHP.
- Every PHP-FPM, Apache and PHP worker was restarted after the update, and every container image was rebuilt from a patched base and redeployed.
- The version was checked through the web server, not only the command line.
- Logs from 17 October onwards were reviewed.
- The host check (lattice-scan --cve …) was run on each server after patching and its output kept with the change record.
- No system older than 8.2 remains internet-facing without a dated exception and an upgrade plan.
- Temporary workarounds that cost functionality have been removed; intentional hardening (upload progress off) is documented.
- Hosts that were exposed to PHP2Shell before patching were checked for webshells and for root activity that no administrator owns.

Sources

- PHP security advisory GHSA-w8hw-w9fx-j79w — CVE-2026-109618, RCE via OOM corruption: https://github.com/php/php-src/security/advisories/GHSA-w8hw-w9fx-j79w
- PHP security advisory GHSA-6wmf-4rv5-p3rp — CVE-2026-109619, RCE via stale global state: https://github.com/php/php-src/security/advisories/GHSA-6wmf-4rv5-p3rp
- CISA alert — Critical Vulnerabilities in PHP Require Immediate Patching (21 October 2026): https://www.cisa.gov/news-events/alerts/2026/10/21/critical-vulnerabilities-php-require-immediate-patching
- CISA Known Exploited Vulnerabilities catalog: https://www.cisa.gov/known-exploited-vulnerabilities-catalog
- Google Cloud — Responding to core PHP vulnerabilities CVE-2026-109618 and CVE-2026-109619: https://cloud.google.com/blog/products/identity-security/responding-to-core-php-vulnerabilities-cve-2026-109618-cve-2026-109619
- Tenable — php2shell frequently asked questions (exploitation timeline, versions, detection): https://www.tenable.com/blog/php2shell-cve-2026-109618-cve-2026-109619-frequently-asked-questions-about-critical-php-remote
- Sansec — PHP2Shell research and 22 October update: https://sansec.io/research/php2shell
- Fortinet PSIRT FG-IR-26-186: https://fortiguard.fortinet.com/psirt/FG-IR-26-186
- F5 — K000163821, PHP vulnerabilities CVE-2026-109618 and CVE-2026-109619: https://my.f5.com/manage/s/article/K000163821