

PHP2Shell (CVE-2026-109618, CVE-2026-109619): Patching the Critical PHP RCE Across Your Fleet

Two critical PHP remote code execution flaws, CVSS 9.2, leaked ahead of the 21 October 2026 fix. What is confirmed, which PHP versions get a patch, what your distro ships, and how to prove every server is fixed.

AUTHOR

PatchMon Team

PUBLISHED

21 October 2026

CATEGORIES

Security, Patch Management

READ ONLINE

<https://patchmon.net/blog/php2shell-cve-2026-109618-cve-2026-109619-php-rce-patch>

Contents

1. [The Short Version](#)
2. [What PHP2Shell Actually Is](#)
3. [Why This One Hurts More Than the Usual PHP CVE](#)
4. [Which PHP Versions Will Get a Fix](#)
5. [What Your Distro Is Shipping](#)
6. [If You Cannot Patch Right Now](#)
7. [How to Tell Your Fleet Is Actually Patched](#)
8. [Bottom Line](#)
9. [Sources and Further Reading](#)

Most security releases arrive unannounced. This one came with a countdown.

On 20 October the PHP security team posted an advance notice: a release on **Wednesday 21 October 2026 at 14:00 UTC**, fixing two critical vulnerabilities, both rated CVSS v4.0 9.2. Within hours the details that were meant to stay under embargo had leaked, a screenshot of an exploit tool called **PHP2Shell** was circulating, and the two bugs had CVE numbers: [CVE-2026-109618](#) and [CVE-2026-109619](#) (<https://sansec.io/research/php2shell>). If you run PHP anywhere (WordPress, Magento, Drupal, Laravel, a control panel, or an internal tool nobody remembers deploying), you are reading this with a deadline.

This post is the practical version. What is actually confirmed, what is still rumour, which PHP versions will get a fix and which never will, what your distro ships, and how to prove every server is running the patched PHP rather than assuming it is.

The Short Version

If you only have two minutes:

What it is: Two remote code execution vulnerabilities in PHP itself, not in a framework or plugin. Tracked as [CVE-2026-109618](#) and [CVE-2026-109619](#) (<https://watchtowr.com/intelligence/php-cve-2026-109618-faq/>), nicknamed PHP2Shell after the leaked exploit tool.

Severity: [CVSS v4.0 9.2 for both, rated by the PHP security team](#) (<https://www.php.net/>).

Disclosed: Advance notice and embargo leak on 20 October 2026. Fixed releases scheduled for 21 October 2026, 14:00 UTC (16:00 CEST, 10:00 EDT).

Who is affected: The official affected range comes with the release. The leaked tool claims every PHP release from 5.4.0 onwards. Treat every PHP install you own as in scope until proven otherwise.

What an attacker gets: Code execution on your web server, as the PHP worker user, over the network.

What you do about it: Update PHP the moment your distro or upstream ships the fix, then **restart PHP-FPM or Apache**. A patched package on disk does nothing while the old worker processes are still running.

The rest of this post is the why and the how.

What PHP2Shell Actually Is

Here is what is confirmed, and here is the line where confirmation stops.

Confirmed. The PHP project's own [advance notice on php.net](#) (<https://www.php.net/>) states that the release "addresses two critical vulnerabilities, both rated CVSS v4.0 9.2", and that the team reserves the right to ship earlier "should details of either vulnerability become public, or should evidence of active exploitation appear in the wild." [watchTower confirmed](#)

(<https://watchtowr.com/intelligence/php-cve-2026-109618-faq/>) that both are remote code execution bugs in PHP core and, per [Huntress](https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities) (<https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities>), reproduced CVE-2026-109618. [Sansec reports](https://sansec.io/research/php2shell) (<https://sansec.io/research/php2shell>) that a screenshot of an exploit tool named PHP2Shell, credited to Calif.IO, leaked alongside the advisory, and that its banner claims it works against every PHP release from 5.4.0 on.

Not yet confirmed. The PHP project has not published the technical mechanism, the affected version list, or the GitHub security advisories. Those land with the release. [Phoenix Security's analysis](https://phoenix.security/php-core-rce-cve-2026-109618-109619-critical-remote-code-execution/) (<https://phoenix.security/php-core-rce-cve-2026-109618-109619-critical-remote-code-execution/>) describes two heap use-after-free bugs, one in the session upload progress handler and one in the `multipart/form-data` request parser, but that description comes from a leaked proof-of-concept bundle that Phoenix itself warns may be hostile, and it notes the path from use-after-free to code execution is asserted rather than demonstrated. Treat it as a plausible lead, not a fact.

The detail that matters most for planning is the one every source agrees on: these bugs live in PHP, not in your application. Sansec puts it bluntly: "You cannot fix it from within your store codebase." No plugin update, no WordPress core release and no application firewall rule inside your app will close them. The fix is a new PHP binary.

Why This One Hurts More Than the Usual PHP CVE

PHP ships security releases most months. [The 24 September release](https://www.php.net/ChangeLog-8.php) (<https://www.php.net/ChangeLog-8.php>) alone fixed eleven advisories across FPM, OpenSSL, SOAP, Phar and the HTTP stream wrapper, and most teams applied it on their normal cycle without drama. So why does this one deserve a dedicated post?

Three reasons.

It is remote and pre-application. Most PHP CVEs need a particular extension, a particular function called with attacker-controlled input, or a particular configuration. If the leaked descriptions are right, these bugs sit in code that runs on incoming requests before your application gets control. [Sansec notes](https://sansec.io/research/php2shell) (<https://sansec.io/research/php2shell>) that "the attack targets PHP itself, before your store code and Shield run." That is the difference between "some apps are exposed" and "every PHP web server is a candidate."

The clock started before the patch did. An embargo leak, a circulating tool screenshot, and an exploit video that was [briefly posted and then removed](https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities) (<https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities>) mean attackers already know where to look. Once the fix is public, the diff between the old and new source shows them exactly where. Sansec says it expects mass attacks "within 15 minutes of the release." Whether it is fifteen minutes or fifteen hours, your patch window is measured in hours, not your usual monthly cycle.

A huge share of PHP in production will never get an upstream fix. As of today, [php.net](https://www.php.net/supported-versions.php) supports only 8.2 and later (<https://www.php.net/supported-versions.php>). PHP 8.1 reached end of life on 31 December 2025; 8.0, 7.x and 5.x are long gone. Those branches get no release on 21 October. If the leaked "5.4.0 onwards" range holds, every legacy PHP box you own is exposed with no upstream patch coming, and your only options are a distro backport, a migration, or taking it off the internet.

This is not a "next maintenance window" CVE. It is a "have the update staged before 14:00 UTC" CVE.

Which PHP Versions Will Get a Fix

The PHP project has said the fixed versions will be published with the release. Based on the current supported branches and their [latest releases](https://www.php.net/supported-versions.php) (<https://www.php.net/supported-versions.php>), this is where things stand:

Branch	Current release (24 Sep 2026)	Upstream fix expected	Security support ends
PHP 8.5	8.5.11	8.5.12	31 Dec 2029
PHP 8.4	8.4.26	8.4.27	31 Dec 2028
PHP 8.3	8.3.35	8.3.36	31 Dec 2027
PHP 8.2	8.2.34	8.2.35	31 Dec 2026
PHP 8.1 and older	n/a	None	Already ended

The "expected" column is the next patch release on each branch, which several security vendors have also cited; confirm against the [php.net release announcement](https://www.php.net/archive/2026.php) (<https://www.php.net/archive/2026.php>) once it is live. PHP 8.2 deserves a note of its own: it gets this fix, but its security support ends on 31 December 2026, ten weeks from now. If you are patching 8.2 hosts this week, put the 8.3 or 8.4 migration on the calendar while you are at it.

If you are on PHP 8.1 or older, read the next section carefully, because your fix (if there is one) comes from your distro, not from php.net.

What Your Distro Is Shipping

Most Linux servers do not run PHP from php.net. They run whatever their distribution packages, and distributions **backport security fixes without changing the upstream version number**. That has two consequences. A distro PHP that reports `8.3.6` can be fully patched.

And a version check against the table above tells you nothing on a distro build: you need the distro's own advisory or package changelog.

Here is what the main distributions ship today, checked against the [Debian archive](https://sources.debian.org/src/php-defaults/) (<https://sources.debian.org/src/php-defaults/>) and [Launchpad](https://launchpad.net/ubuntu/+source/php-defaults) (<https://launchpad.net/ubuntu/+source/php-defaults>):

Distribution	Default PHP	Upstream branch status	Where the fix comes from
Ubuntu 26.04 LTS	8.5.4	Supported	Ubuntu security update
Ubuntu 24.04 LTS	8.3.6	Supported	Ubuntu security update
Ubuntu 22.04 LTS	8.1.2	Upstream EOL	Ubuntu backport only
Debian 13 "trixie"	8.4.24	Supported	Debian security update (DSA)
Debian 12 "bookworm"	8.2.32	Supported to Dec 2026	Debian security update (DSA)
Debian 11 "bullseye"	7.4.33	Upstream EOL	None. Debian LTS ended 31 August 2026

Ubuntu. The version strings make the point: 22.04 still reports PHP 8.1.2 and 24.04 reports 8.3.6, but both carry years of Ubuntu security revisions. Watch the [Ubuntu CVE tracker](https://ubuntu.com/security/cves) (<https://ubuntu.com/security/cves>) for CVE-2026-109618 and CVE-2026-109619. 22.04 matters most: upstream will not ship an 8.1 fix, so a Canonical backport is the only route.

Debian. Debian 12 and 13 are covered through normal security channels; follow the [Debian security tracker](https://security-tracker.debian.org/tracker/) (<https://security-tracker.debian.org/tracker/>). Debian 11 is the uncomfortable one. Its LTS window [closed on 31 August](https://patchmon.net/blog/debian-11-openssl-3-end-of-life) (<https://patchmon.net/blog/debian-11-openssl-3-end-of-life>), so a bullseye host running PHP 7.4 has no free patch coming from anyone. If you still have one serving web traffic, this week is when that stops being a backlog item.

RHEL, Rocky, AlmaLinux. PHP comes from AppStream module streams, so different hosts on the same major release can run different PHP branches. Run `dnf module list php` to see which stream each host is enabled on, and watch for the RHSA and the matching Rocky and Alma errata. If you use Remi's repository, you will typically get the upstream release directly, with an upstream version number.

Alpine and containers. Alpine packages each PHP branch separately (`php82` , `php83` , `php84` and so on), and there is already an [aports work item](https://gitlab.alpinelinux.org/alpine/aports/-/work_items/18512) (https://gitlab.alpinelinux.org/alpine/aports/-/work_items/18512) tracking this. If you build from the official `php` Docker images, the fix arrives as rebuilt images: pull them, rebuild anything that uses them as a base, and redeploy. A running container does not pick up a new image on its own.

Managed hosting and control panels. cPanel, Plesk and most managed hosts ship their own PHP builds. [Sansec's advice](https://sansec.io/research/php2shell) (<https://sansec.io/research/php2shell>) is to ask your provider today to stand by and apply the update as soon as it is released.

If You Cannot Patch Right Now

Before the fix ships, and for any host that cannot take it immediately, there are a few things worth doing. None of them is a substitute for the update.

Find every PHP install first. Hosts often run more than one PHP version side by side, especially where a control panel or Remi repository is involved.

```
# Debian / Ubuntu: every installed PHP package and its version
dpkg -l 'php*' | awk '/^ii/ {print $2, $3}'

# RHEL / Rocky / Alma: same, plus the enabled module stream
rpm -qa 'php*' --qf '%{NAME} %{VERSION}-%{RELEASE}\n'
dnf module list --enabled php

# Every PHP-FPM master currently running, and when it started
ps -eo pid,lstart,cmd | grep '[p]hp-fpm: master'
```

Reduce exposure on legacy hosts. For anything on PHP 8.1 or older with no backport coming, [Huntress recommends](https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities) (<https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities>), upgrading, taking it off the internet, or putting it behind a VPN or web application firewall until you can.

Consider disabling session upload progress. If the leaked description is accurate, one of the two bugs is reached through PHP's session upload progress feature. Turning it off is cheap and rarely breaks anything, since few applications use it:

```
; php.ini, or a drop-in under conf.d / mods-available
session.upload_progress.enabled = Off
```

Restart PHP-FPM or Apache afterwards. Be clear about what this is: a partial measure aimed at one *suspected* path, based on unconfirmed details. It does nothing for the second bug and may turn out to do nothing for the first.

Watch what PHP spawns. A successful exploit usually ends with the PHP worker launching a shell. Huntress suggests alerting on PHP-FPM, php-cgi or Apache processes starting shells, interpreters or download tools, and on commands such as `whoami` or `id` running as the web server user. A quick manual look:

```
# Anything a PHP-FPM worker has spawned
for p in $(pgrep -f '^php-fpm: pool'); do ps --no-headers -o pid,user,lstart,cmd -
-ppid "$p"; done
```

On a healthy PHP web server this usually prints nothing.

How to Tell Your Fleet Is Actually Patched

PHP has the same trap as a kernel update, in a smaller form. With the [Copy Fail kernel bug](https://patchmon.net/blog/copy-fail-cve-2026-31431-linux-kernel-patch) (<https://patchmon.net/blog/copy-fail-cve-2026-31431-linux-kernel-patch>) the problem was a patched kernel on disk and an unpatched one running until reboot. With PHP it is a patched package on disk and the old PHP-FPM workers still serving requests until someone restarts the service. Three things tend to go wrong:

The package updated, the service did not restart. Some packaging restarts PHP-FPM on upgrade and some does not, and a host running two PHP versions may only restart one. Compare the start time of each PHP-FPM master (from the `ps` command above) with the time the package was installed.

PHP is somewhere you did not look. The second PHP version a control panel installed. The bullseye box behind an internal tool. The container image built six months ago from `php:8.1-fpm`. The appliance VM with a web UI.

You cannot tell a patched distro build from an unpatched one by version. `8.3.6` on Ubuntu 24.04 says nothing on its own. Only the package changelog does:

```
# Debian / Ubuntu
apt changelog php8.3-fpm 2>/dev/null | grep -E 'CVE-2026-10961(8|9)'

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

Running that across one server is easy. Running it across two hundred, and keeping the answer current while distro updates trickle in over the next few days, is the actual job.

This is the problem [PatchMon](https://patchmon.net/features/patch-management) (<https://patchmon.net/features/patch-management>) is built for. Each PatchMon agent reports every installed package, its version, and the updates available for it, including whether an available update is a security update. Search the Packages view for `php` and you get every host carrying a PHP package, the version each one has installed, and which of them still has an update waiting. Filter to hosts with pending updates and that is your outstanding list; as each host updates and reports back, it drops off. For containers, the [Docker integration](https://patchmon.net/features/docker-monitoring) (<https://patchmon.net/features/docker-monitoring>) flags containers whose image has an update available, which is how you catch the stale `php:*-fpm` images. And every update is timestamped, which is the evidence your auditor or customer will ask for once the dust settles.

The quickest way to see it on your own servers is a [PatchMon Cloud trial](https://patchmon.net/pricing) (<https://patchmon.net/pricing>): 14 days, billed on hosts that actually check in, nothing to pay if you cancel before day 14. If you would rather self-host, the [Community Edition](https://patchmon.net/open-source) (<https://patchmon.net/open-source>) is AGPLv3 and free, and runs on your own infrastructure in under half an hour.

Bottom Line

PHP2Shell is two critical, pre-application remote code execution bugs in PHP itself, with an exploit tool already shown publicly and a fix landing at 14:00 UTC on 21 October. The part you control is how quickly every PHP process you own is running the fixed code.

The thing to focus on today:

Inventory every PHP install now, before the release, including second PHP versions, containers and anything on PHP 8.1 or older.

Apply the update as soon as your source ships it: php.net for upstream builds, your distro's security feed for packaged PHP, rebuilt images for containers.

Restart PHP-FPM or Apache on every host. A patched package on disk does nothing until the workers restart.

Verify by changelog, not by version string, and keep one report covering the whole fleet rather than spot-checking over SSH.

Isolate or retire legacy PHP. Anything on 8.1 or older without a distro backport should come off the public internet this week.

The teams that get through events like this calmly are the ones who can answer "which servers still run an unpatched PHP?" in minutes, on the day the question is asked. Whatever you use to get that answer, today is a good day to check that it works. If you want to see how PatchMon handles it, the [14-day Cloud trial](https://patchmon.net/pricing) (<https://patchmon.net/pricing>) is the quickest way in. Either way, get PHP patched first.

Sources and Further Reading

[php.net: Upcoming Critical PHP Security Release \(advance notice\)](https://www.php.net/) (<https://www.php.net/>)

[php.net: Supported Versions](https://www.php.net/supported-versions.php) (<https://www.php.net/supported-versions.php>)

[php.net: PHP 8 ChangeLog](https://www.php.net/ChangeLog-8.php) (<https://www.php.net/ChangeLog-8.php>)

[php-src GitHub security advisories](https://github.com/php/php-src/security/advisories) (<https://github.com/php/php-src/security/advisories>)

[Sansec: Critical PHP RCE, CVE-2026-109618 and CVE-2026-109619](https://sansec.io/research/php2shell)

(<https://sansec.io/research/php2shell>)

[watchTower: PHP CVE-2026-109618 FAQ](https://watchtower.com/intelligence/php-cve-2026-109618-faq/) (<https://watchtower.com/intelligence/php-cve-2026-109618-faq/>)

Huntress: PHP2Shell, leaked critical PHP vulnerabilities (<https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities>)

Phoenix Security: PHP core RCE, CVE-2026-109618 and CVE-2026-109619
(<https://phoenix.security/php-core-rce-cve-2026-109618-109619-critical-remote-code-execution/>)

Decipher: Researchers warn of upcoming critical PHP flaws
(<https://decipher.sc/2026/10/20/researchers-warn-of-upcoming-critical-php-flaws/>)

Alpine aports: PHP security release tracking
(https://gitlab.alpinelinux.org/alpine/aports/-/work_items/18512)

Debian security tracker (<https://security-tracker.debian.org/tracker/>)

Ubuntu CVE reports (<https://ubuntu.com/security/cves>)

endoflife.date: PHP (<https://endoflife.date/php>)

The open source Linux patch management platform

PatchMon gives sysadmins one dashboard for patching across Linux and FreeBSD fleets, with Windows supported alongside. Run it as a managed SaaS on PatchMon Cloud (per-host billing, 14-day trial, no fleet minimum) or self-host the AGPLv3 Community Edition on your own infrastructure.

[Start a trial: patchmon.net/pricing](https://patchmon.net/pricing)

