

PHP2Shell: The Patch Is Coming and the Mitigation Is Dead

TechRounder PDF Edition

Live article: <https://www.techrounder.com/security/php2shell-critical-php-flaw-patch-confusion/>

By Vipin PG | Published October 22, 2026 | Updated October 22, 2026 | Format: Deep Dive | 9 min read

In brief

A critical bug in the runtime behind most of the web forced an emergency patch, a broken embargo and a retracted fix. Getting the update installed is where the real work starts.

A critical bug in the runtime behind most of the web forced an emergency patch, a broken embargo and a retracted fix. Getting the update installed is where the real work starts.

On the morning of 21 October, a system administrator somewhere read a one-line mitigation for the PHP vulnerability everyone was talking about, applied it, and felt better. Turn off `session.upload_progress.enabled` in `php.ini`. It takes seconds, breaks almost nothing visible, and had spread through Slack channels and vendor notes faster than any patch could.

Then the company that proposed it took it back. GMO Flatt Security, whose write-up is the most careful technical account of the bug, updated its own post to say the setting "may help, but attacks are still possible, and this is only a partial mitigation." That update landed at 05:00 UTC, roughly nine hours before PHP's scheduled release and a few hours after the advice had already reached production.

The flaw and the patch are real. Much of what circulated around them was premature, contradictory or later withdrawn.

What PHP actually announced

On 20 October, the PHP development team published an advance notice on `php.net`. It said two critical vulnerabilities had been found, that the release would happen on Wednesday 21 October at 14:00 UTC, and that the team reserved the right to ship earlier if details leaked or exploitation appeared.

It gave no CVE identifier, component, affected versions or fixed versions. The forward to the `oss-security` list was explicit that the details would come "with the release." The PHP Foundation's forum copy carried the same text and, by midday on 21 October, had drawn about 2,400 views.

Both flaws were rated 9.2 under CVSS v4.0, the scoring system that ranks severity out of ten. A score without a published vector string, the breakdown of attack path, complexity, privileges and impact, cannot be recomputed or audited, which is why some researchers would not treat it as settled.

Pre-announcing a security release is unusual for PHP, and not automatically a mistake. It gave hosting providers time to inventory versions and book change windows, where a silent release would have left everyone reacting in the same hour.

How the embargo broke, and why the date moved

The advance notice was supposed to be the only thing anyone knew until Thursday 22 October. Instead, a "Tell HN" post named both identifiers, claimed a 10.0 severity and repeated the Thursday date. That thread reached 1,165 points and 471 comments.

watchTowr, a security firm tracking the disclosure, confirmed the bugs were real and under embargo. PHP then used its own escape hatch and moved the release forward a day. That compressed every defender's preparation window rather than extending it.

The leak got some details right and others wrong. The leaked score was 10.0; the official one was 9.2. The leaked date was Thursday; the release arrived Wednesday. Anyone who built a plan on the first numbers had to rebuild it within hours.

The identifiers nobody had registered

By midday on 21 October, two CVE identifiers, CVE-2026-109618 and CVE-2026-109619, were in headlines, scanner signatures and social posts worldwide. Neither existed in the databases that make a CVE mean something.

NVD, the National Vulnerability Database, returned `totalResults: 0` for both through its API. MITRE's CVE Services returned `CVE_RECORD_DNE`, meaning the record does not exist. CISA's Known Exploited Vulnerabilities catalogue, at version 2026.10.20, listed neither. The identifiers had been reserved out of band; the records had not been published.

Ticketing systems, SBOM matchers and scanners key on CVE identifiers. If the patch ships before the record does, the tools meant to tell you that you are exposed will stay quiet while the fix sits in a repository. Track the PHP version this week, not the CVE number.

What the flaw appears to be

PHP has not published a root-cause analysis. The most detailed description comes from a GitHub repository that admits it has no working exploit, and from GMO Flatt's independent testing. Both point at the same place, which is the strongest signal available before the advisory.

The claim is a use-after-free in PHP's session extension, in the code that handles file upload progress. A use-after-free is a memory-safety bug: a program frees a block of memory, keeps a pointer to it, and later writes through that pointer. If an attacker controls what gets allocated into that freed space, a crash can sometimes become code execution.

The repository describes the mechanism inside `php_session_rfc1867_callback()`, the handler for RFC1867-style multipart uploads, the ordinary browser file-upload format. It stores raw pointers to values inside a hash table. When progress is updated, the table is rewritten, the old value destroyed, and the stored pointers left dangling. The exploit reclaims that memory with attacker-shaped structures so a later internal call executes a system command.

The corroboration is a screenshot from the researcher showing heap addresses, a stale reference, and a shell prompt returning `uid=33(www-data)` after a 22.6-second chain. That evidence covers one configuration: Ubuntu 26.04, PHP 8.5.4, Apache with `mod_php`. It does not cover every PHP installation on earth, and treating it as if it does is how a bug becomes a panic.

If the description holds, exposure is wider than the application code suggests. The handler runs while the request is parsed, before your scripts execute, so an application that never calls `session_start()` can still reach it. The PHP manual has also warned for years that upload progress writes client-supplied file and field names into the session store without validation.

The mitigation that died in a morning

Disabling `session.upload_progress.enabled` spread first because it was cheap and plausible. The setting defaults to on, so the change applies to nearly every stock install, and it targets the feature named in the analysis.

GMO Flatt's retraction is the part that matters. Its post now says the change may help but does not close the path, and that a patch is fundamentally required. Two claims cannot both be right: the repository that popularised the workaround calls the path neutralised, while the firm that proposed it says partial. Until the advisory lands, treat it as partial.

Sansec's recommendation is more aggressive: block multipart/form-data at the web server or edge, because parsing happens before application code runs. That is technically coherent and operationally expensive. Blocking every multipart POST breaks file upload, avatar changes, CSV imports and a long list of admin panels. If you do it, scope it by path, and expect to hear about it within the hour.

Patch first, configure second. A `php.ini` line that survives in a runbook after the patch is applied is a habit, not a control.

Who is exposed, and how much of the web that is

PHP runs on about 69.3% of websites, according to W3Techs. PHP 8 accounts for 45.2% of all sites, about 65.2% of PHP sites, a distinction that got flattened in several early reports. Within PHP 8, the largest single block is 8.3 at 34.8%, followed by 8.2 at 27.7% and 8.1 at 15.2%.

Those are shares of websites, weighted by popularity. A separate scan reported by SecurityWeek counts responding hosts: roughly 1.5 million internet-facing PHP servers, of which 883,717 (59%) run 8.1 or earlier. Another 740,000 hide their version entirely, so their operators have to inventory from the inside.

The two datasets measure different things and should not be divided into each other. They agree on the shape of the tail: a large installed base on branches that will not receive a fix.

Who gets a patch, and who is told to cope

PHP's support policy is published, and it draws a hard line. Four branches are expected to receive fixes. Everything below 8.2 is end-of-life, which means no upstream patch at all.

Image: PHP branch support status and pre-release fix predictions, 21 October 2026

Source: `php.net` supported versions, with predicted fixed versions from the host scan. Those numbers are pre-release predictions and may be wrong.

Two caveats matter in practice. Linux distributions backport fixes, so Debian and RHEL users will see vendor version strings rather than 8.2.35, and their patch date is the distribution's. Most of the 69% also does not run upstream PHP at all. It runs a hosting panel's build, a managed platform's build, or a bundled interpreter inside an appliance, and for those operators the real patch date is whenever the vendor ships.

For PHP 5, at roughly 7.7% of PHP sites, there is no patch coming and no honest way to imply otherwise. The options are isolation, blocking at the edge, or migration. Commercial extended-support vendors will cover some of the 7.x tail. Neither group should be told to "apply the update."

The edge: eight rules and a disclosure gap

Cloudflare shipped two emergency WAF releases on 21 October: four rules in its managed ruleset, four in its free ruleset, all set to block, all named "Generic Rules - Anomalous Execution". The changelog describes the free release as adding two rules in prose while listing four in its table, which tells you something about the hour it was written in.

Neither entry names a CVE. The generic rule names are consistent with embargo hygiene and unhelpful if you want to verify coverage. Anyone outside Cloudflare cannot see what those rules match, so "we're covered by Cloudflare" is currently a statement about a vendor's private detection logic, and you cannot test it. The same organisation's chief technology officer is also the source of the early-attack claim. That is worth naming as a conflict of interest, though it is no reason to dismiss the claim.

Is anyone actually attacking?

The reports contradict each other, and the contradiction stands. Sansec, relaying Cloudflare, reported early attacks at 07:43 UTC on 21 October. watchTower's FAQ, updated the same day, said exploitation was "none reported" and CISA KEV "not listed." Huntress reported no in-the-wild exploitation and no public proof of concept at publication. Hadrian's alert carries an "exploited in the wild" badge that conflicts with both.

The likely explanation is terminology. "Attacks" can mean scanning or blocked attempts at the edge, while "exploitation" usually means a confirmed compromise on a monitored host. Those are different observations, made by parties with different visibility. "No confirmed exploitation" reports an absence of evidence and does not show that anyone is safe.

The evidence base is thin in other ways too. For much of the day, the top search result for the CVE was a repository with 82 stars, seven commits and no exploit, promising code "very soon."

What would change this story

The advisory itself is the test. A narrower affected range than "5.4 and above" would invalidate the end-of-life panic. A published CVE record with a real vector string would either confirm the 9.2 or deflate it. Evidence that the session upload-progress path is not the root cause would invalidate the entire mitigation discussion above.

The sceptics have a point. "Unpatchable third of the web" overstates it, since a branch without an upstream patch can still be defended through edge blocking, network isolation, vendor backports and extended support. And memory corruption in a long-standing feature does not mean nobody looked. The manual documented the dangerous behaviour years ago.

What to do now, and what to watch

- Inventory before you patch. You cannot fix a version you have not found. Include containers, build images, appliances and desktop bundles, not just the fleet your configuration management can see.
- Patch supported branches first, then work downwards. For 8.1 and below the decision is a migration or a compensating control, and it needs an owner. "We'll get to it" is how a tail becomes permanent.
- Treat configuration changes as temporary. If you disabled upload progress, log it, set a review date, and remove it once the patched build is in place.
- Watch whether the CVE records land with the release, whether your distribution ships a backport first, whether a hosting panel ships out of band, and whether the early-attack claim resolves into blocked scanning or a confirmed compromise. The first three decide your patch date. The fourth decides your urgency.

PHP did not handle this badly. Pre-announcing gave defenders hours they would not otherwise have had. What went wrong was the gap between warning and patch, which filled with confident answers: a severity score without a vector, a CVE without a record, a mitigation without its retraction attached. The teams that came through the week well treated all three as provisional.

References

1. php.net - archive / 2026.php - <https://www.php.net/archive/2026.php#2026-10-20-1>
2. seclists.org - oss-sec / 2026 - <https://seclists.org/oss-sec/2026/q4/266>
3. discourse.thephp.foundation - t / announce-upcoming-critical-php-security-release - <https://discourse.thephp.foundation/t/announce-upcoming-critical-php-security-release/6251>
4. news.ycombinator.com - item - <https://news.ycombinator.com/item?id=50172041>
5. blog.flatt.tech - entry / php2shell - <https://blog.flatt.tech/entry/php2shell>
6. w3techs.com - technologies / details - <https://w3techs.com/technologies/details/pl-php>
7. securityweek.com - php2shell-mass-exploitation-looms-for-leaked-critical-php-flaws - <https://www.securityweek.com/php2shell-mass-exploitation-looms-for-leaked-critical-php-flaws/>
8. php.net - supported-versions.php - <https://www.php.net/supported-versions.php>
9. developers.cloudflare.com - waf / change-log - <https://developers.cloudflare.com/waf/change-log/changelog/>
10. sansec.io - research / php2shell - <https://sansec.io/research/php2shell>
11. watchtowr.com - intelligence / php-cve-2026-109618-faq - <https://watchtowr.com/intelligence/php-cve-2026-109618-faq/>
12. huntress.com - blog / php2shell-leaked-critical-php-vulnerabilities - <https://www.huntress.com/blog/php2shell-leaked-critical-php-vulnerabilities>