

October 21, 2026

PHP2Shell: Pre-Authentication Remote Code Execution and Memory Disclosure Flaws in Core PHP Engine (CVE-2026-109618 & CVE-2026-109619)

Key Insights:

- CVE-2026-109618 and CVE-2026-109619 (collectively tracked as PHP2Shell) are critical (CVSS 9.2) memory-safety vulnerabilities in the core PHP engine that allow unauthenticated remote code execution and sensitive process memory disclosure.
- Because PHP powers approximately 70% of all websites globally, virtually all web platforms—including WordPress, Drupal, Joomla, Magento and custom enterprise tools—are exposed regardless of their application-level security.
- Although the underlying bugs have existed in the PHP codebase for over a decade (affecting versions back to PHP 5.4), modern frontier AI models have eliminated the complexity barrier required to chain and adapt these memory flaws across differing server environments.
- Vulnerability details were leaked online on October 20, ahead of the scheduled patch release, compressing the remediation window as threat actors actively scan the internet and use AI to automate exploit creation.
- The only complete remediation is upgrading to PHP 8.5.12, 8.4.27, 8.3.36, or 8.2.35, while deploying Radware Web Application Firewall (WAF) signatures through Security Update Services (SUS) and applying runtime configuration hardening for immediate interim protection.

The newly disclosed PHP2Shell vulnerabilities (CVE-2026-109618 and CVE-2026-109619) present a critical, CVSS 9.2 threat to the global web infrastructure, allowing unauthenticated remote code execution and sensitive memory disclosure within the core PHP engine. Because PHP powers approximately 70% of all websites, nearly all platforms and custom applications are exposed at the engine level, regardless of their specific application logic or security posture. Compounding this risk, vulnerability details leaked ahead of the coordinated patch release, granting threat actors a crucial head start to leverage AI-assisted automated exploit generation against a massive attack surface before organizations could deploy defenses.

Immediate mitigation requires a defense-in-depth approach, prioritizing the deployment of upstream engine patches (PHP versions 8.5.12, 8.4.27, 8.3.36, or 8.2.35) as the only permanent remediation. For organizations unable to patch immediately, interim protection must be established by deploying Radware Web Application Firewall (WAF) signatures via Security Update Services (SUS) to intercept malformed multipart requests and block incoming exploit

payloads. This perimeter defense should be actively reinforced with strict runtime configuration hardening—such as disabling session upload progress and enforcing strict memory limits—to isolate and protect active PHP worker processes until patches are fully deployed.

Overview and Timeline of Events

On the evening of October 20, 2026, details and CVE identifiers for two critical vulnerabilities in the core PHP scripting engine were leaked on the Hacker News forum ahead of their scheduled coordinated disclosure. Tracked as CVE-2026-109618 and CVE-2026-109619, and collectively referred to as PHP2Shell, the flaws carry a CVSS severity rating of 9.2. Both vulnerabilities allow unauthenticated attackers to execute arbitrary code remotely or extract sensitive data directly from web server memory without requiring a password or user session.

The premature leak eliminated the lead time defenders typically rely on to prepare patches before threat actors become aware of a flaw. Throughout the evening of October 20 and the early morning of October 21, cybersecurity authorities—including [CERT-EU](#) and national agencies in the [United Kingdom](#), [Germany](#), [the Netherlands](#), Japan and Australia—issued urgent warnings advising administrators to prepare for emergency patching. CERT-EU specifically advised organizations unable to patch immediately to consider taking vulnerable PHP websites offline. Following this guidance, hundreds of websites, including several government portals and commercial services from organizations such as [Nintendo](#), temporarily suspended operations prior to the patch release.

On October 21, 2026, at approximately 13:35 UTC (ahead of the planned 14:00 UTC schedule), the PHP development team released emergency patches in versions 8.5.12, 8.4.27, 8.3.36, and 8.2.35. Given the severity of the threat, the maintainers also provided minimal source-code backports for end-of-life branches, specifically PHP 7.4, 8.0, and 8.1. Shortly after the patches became public, proof-of-concept exploits began circulating online. Security researchers demonstrated a combined exploit script capable of achieving unauthenticated remote code execution against the latest default container images of WordPress, Drupal and Joomla.

Risk Assessment: Attack Surface and the AI Inflection Point

[PHP underpins an estimated 70% of all websites on the internet](#), serving as the foundation for major content management systems like WordPress, Drupal, Joomla, Magento, and Wikipedia's MediaWiki, as well as countless internal enterprise applications handling customer data and business logic. Because the PHP2Shell vulnerabilities reside in the core PHP engine itself rather than in any specific application code, every PHP installation serving web requests is at risk. This includes environments running PHP-FPM, Apache with `mod_php`, `php-cgi`, FrankenPHP and LiteSpeed. Even a basic "Hello World" script running on an affected PHP version is vulnerable. The PHP security team assesses the risk to be highest for Apache `mod_php` deployments, where an unauthenticated request can trigger the exploit before any application code is executed.

Furthermore, millions of websites globally continue to run legacy PHP versions that are no longer actively supported and cannot be easily updated. Nearly every PHP release from the last decade, dating back to PHP 5.4, is affected by at least one of these flaws.

The discovery and weaponization of PHP2Shell also mark a shift in the economics of vulnerability exploitation. The underlying memory-safety flaws had remained hidden in the PHP codebase for over a decade because turning them into working exploits required chaining multiple complex bugs and accounting for exact memory allocations across requests. Historically, differences in PHP versions, server configurations, and memory layouts meant that an exploit built for one target would fail on another without extensive manual reverse engineering and debugging.

During the [research that uncovered PHP2Shell](#), Wiz researchers utilized frontier AI models including Gemini 4 Argon and GPT-5.6 Sol to bridge this gap. According to the researchers, Argon made a particularly significant contribution to finding and helping to remediate the vulnerability. Large language models paired with debugging tools can analyze memory failures, iterate on adjustments, and adapt complex exploit chains to different PHP versions and configurations automatically. Because the flaws leaked early, threat actors may already have cataloged exposed PHP servers by version and are using AI to compare the unpatched and patched versions of the source code, enabling the generation of working exploits in just a few hours.

Technical Deconstruction of PHP2Shell

The PHP2Shell vulnerabilities operate below the application layer during request setup, cleanup, and state management. The flaws are categorized under two primary CVEs encompassing four distinct memory corruption and disclosure primitives, though researchers note that the underlying issues are deeply rooted in the codebase and other variants fold under these same identifiers.

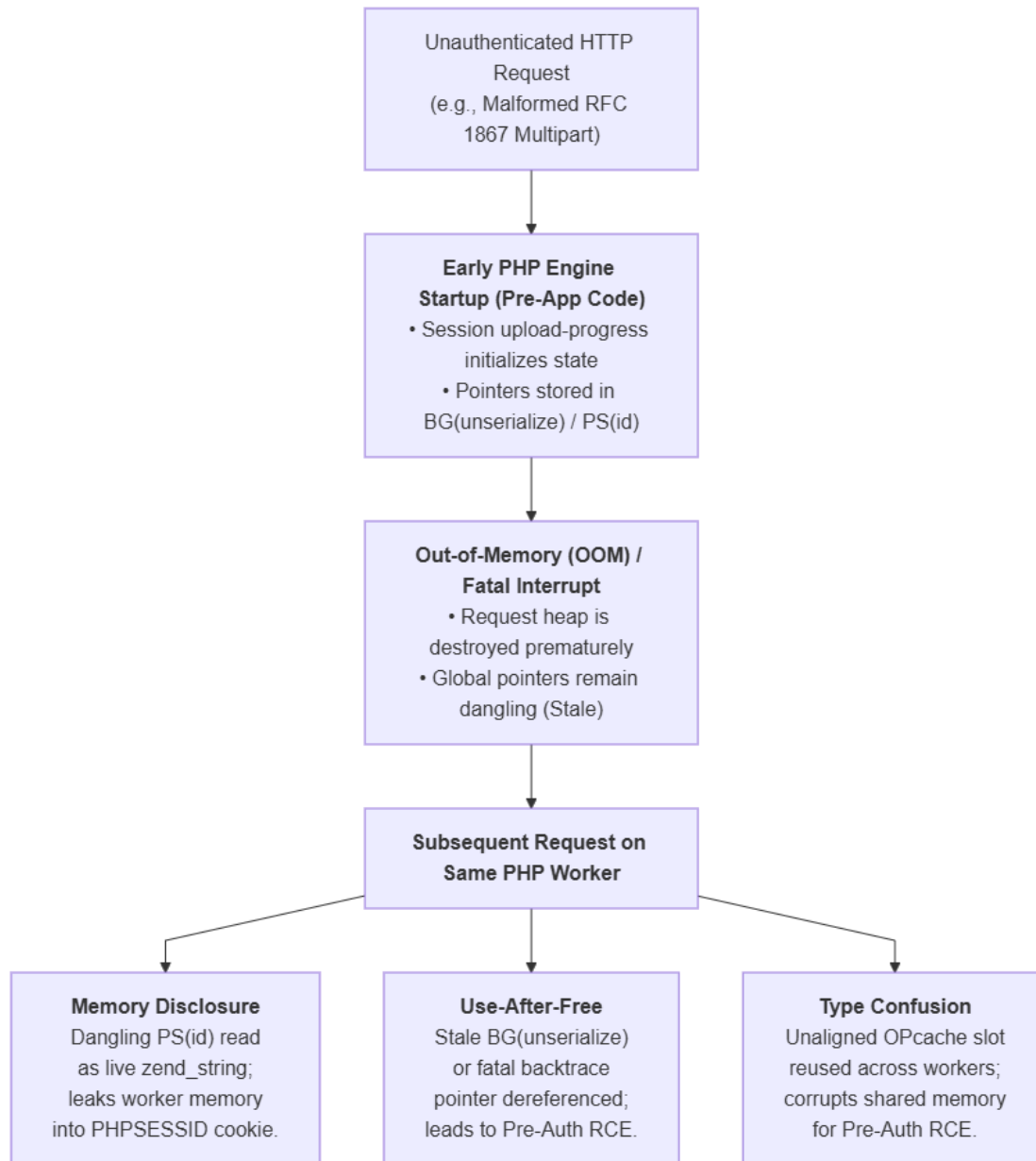


Figure 1: PHP2Shell vulnerabilities

CVE-2026-109618: RCE via Out-of-Memory (OOM) Corruption and Memory Disclosure

CVE-2026-109618 (GHSA-w8hw-w9fx-j79w¹) occurs when out-of-memory handling interrupts PHP while internal request data is only partially initialized, causing the subsequent request

¹ the GitHub Security Advisory identifier for CVE-2026-109618

cleanup routine to treat invalid data as valid. Confirmed vectors to trigger this state reside in RFC 1867 multipart request parsing, which executes during early request startup before normal module initialization completes.

Within this pre-initialization window, PHP's Session module can initialize session state during upload-progress handling and publish a session ID pointer in PS(id). If an out-of-memory error aborts the request before cleanup finishes, PHP destroys the request heap but leaves PS(id) pointing to the freed memory allocation. When a subsequent multipart request is processed by the same PHP worker, it reaches the upload-progress cleanup path before normal session initialization can reset the pointer. PHP trusts the dangling pointer as a live zend_string and URL-encodes the reclaimed heap contents directly into a PHPSESSID response cookie. This allows an attacker to remotely read regions of live worker memory containing passwords, API keys, authentication tokens, and certificates from other requests, even if the hosted application never calls session_start().

In that same pre-initialization window, the session upload-progress handler can also decode session data, storing request-owned bookkeeping in the global BG(unserialize) state. A fatal memory allocation failure interrupts this path, destroys the request heap, and leaves the unserializer's global state published. On a subsequent request to the same worker, PHP consumes attacker-reclaimed memory as live unserializer bookkeeping. By combining this use-after-free primitive with the PHPSESSID memory disclosure to bypass address layout uncertainty, attackers can corrupt process memory and achieve unauthenticated remote code execution.

CVE-2026-109619: RCE via Stale Global State and OPcache Type Confusion

CVE-2026-109619 (GHSA-6wmf-4rv5-p3rp) involves request-owned internal state surviving after its backing memory has been freed, allowing a subsequent request on the same worker to access stale state before normal initialization resets it.

The first primitive under this CVE abuses fatal-error backtraces (specifically affecting PHP 8.5). PHP retains the backtrace of the most recent fatal error in the executor-global variable EG(last_fatal_error_backtrace). When a fatal out-of-memory error occurs, PHP destroys the request heap without undefining that global value, leaving the surviving worker with a pointer to freed memory from the prior request. If that same worker emits an early diagnostic message during the next request, it accesses the dangling pointer before the executor reinitializes its state. PHP then attempts to update or destroy the dangling backtrace using attacker-reclaimed memory, creating a cross-request use-after-free condition that leads to remote code execution.

The second primitive involves OPcache shared-memory type confusion. OPcache stores compiled PHP structures in shared memory and assigns indices into per-worker runtime-pointer maps, which workers coordinate via a shared frontier. During zend_accel_script_persist(), PHP advances its process-local frontier (CG(map_ptr_last)) before publishing the updated shared

frontier upon completion. If a memory allocation failure occurs between these two steps, the process-local advance remains in place while partially persisted objects refer to slots that the shared state still treats as available. When another worker reuses one of those slots for a different runtime object, the stale shared metadata resolves to a pointer of the wrong type, corrupting shared OPcache state and enabling unauthenticated remote code execution.

Mitigation: A Multi-Layered Defense Strategy

Because PHP2Shell executes below the application layer, standard application logs often contain little context of an intrusion. Defenders must rely on a multi-layered approach: deploying perimeter Web Application Firewall (WAF) protections for immediate containment, applying engine-level configuration workarounds where patching is delayed, monitoring server telemetry, and installing the upstream PHP patches for permanent remediation.

Immediate Interim Defense via Radware

While patches are tested and rolled out across production environments, web application and API protection provides an essential first line of defense to inspect incoming HTTP traffic and block exploit payloads before they reach vulnerable PHP workers. Radware's Security Update Service (SUS) provides updated signatures and policies across its Cloud Application Protection, AI Xploit Shield, and on-premises AppWall and Defense Pro protection solutions to detect and mitigate exploit attempts targeting CVE-2026-109618 and CVE-2026-109619. Radware's full-transaction inspection and positive security models enforce strict HTTP compliance, filtering anomalous multipart requests, blocking oversized or malformed header and body structures, and preventing the reflection of leaked process memory in session cookies.

Permanent Remediation: Upgrading PHP

Applying the official PHP security updates is the only complete solution to eliminate these vulnerabilities at their source. Administrators should immediately update to PHP 8.5.12, 8.4.27, 8.3.36, or 8.2.35. Major Linux distributions and long-term support (LTS) vendors received patches in advance and are distributing updated packages.

Organizations relying on the volunteer-maintained rpms.remirepo.net repository are advised by the PHP team to switch to official distribution packages or build from source, as the third-party repository experienced availability delays during the initial release. For legacy environments running unsupported versions (PHP 8.1 and older), administrators should contact their LTS vendor for backported fixes, upgrade to a supported branch, or compile from source using the minimal backports provided by the PHP security team for versions 7.4, 8.0, and 8.1.

Runtime Configuration Workarounds

For environments that cannot be patched immediately, [the PHP team has outlined temporary configuration measures](#) that reduce exposure to specific attack paths, though they do not render

an installation completely safe. All settings must be enforced at the server or FastCGI Process Manager (FPM) pool level (such as via `php.ini` or `php_admin_value` in Apache/FPM configurations) so applications cannot override them; administrators must not rely on `ini_set()` or `.user.ini` files.

To reduce exposure to **CVE-2026-109618 (OOM Corruption)**, administrators should set `memory_limit = -1` and verify that no pool, directory, or application configuration reintroduces a finite limit. While this removes the controllable PHP memory-limit failure, an underlying system allocation failure can still reach the vulnerable code path. To mitigate that risk, enforce a strict `post_max_size` (setting it to 8M for applications that do not require file uploads, or 32M where uploads are necessary) and keep `upload_max_filesize` below `post_max_size`. Additionally, enforce strict limits at the web server, reverse proxy, or WAF level for decoded request size, input field count, multipart part count, header count, and nesting depth, ensuring these rules also apply to requests lacking a valid Content-Length header. Finally, use container or Linux cgroup limits as an availability boundary configured to terminate only the affected worker process upon memory exhaustion; do not use `RLIMIT_AS` or `RLIMIT_DATA`, as these directly trigger the vulnerable allocation failure.

To reduce exposure to **CVE-2026-109619 (Stale Global State)**, administrators should set `session.upload_progress.enabled = 0` and restart all PHP workers, which blocks the confirmed session upload-progress exploitation paths. On PHP 8.5 installations, additionally set `fatal_error_backtraces = 0` and restart every worker to close the fatal backtrace vector. Administrators should also remove any invalid or deprecated per-request settings that emit diagnostic warnings prior to engine initialization (note that simply hiding warnings via `error_reporting`, `display_errors`, or the `@` operator does not prevent the underlying state access) and resolve any attacker-reachable fatal errors or execution timeouts in upload or callback routines.

Detection and Threat Hunting

Until all systems are fully patched, security operations teams should monitor web server and proxy telemetry for indicators of exploitation. Key signals include unusual bursts of multipart HTTP requests followed by memory-limit errors or repeated HTTP 500 internal server responses, unexpectedly large or malformed PHPSESSID cookies in outbound HTTP responses, and unexplained PHP worker process churn or crashes.

Are My Radware Products Affected?

Radware evaluated the impact of CVE-2026-109618 and CVE-2026-109619 on its own product portfolio while actively providing preemptive protection through our cyber defense products and cloud security services. Customers should check [this advisory](#) on the customer portal for updates on all products, services and detailed mitigation instructions.

Radware's security researchers are continuing to monitor in-the-wild exploitation activity and will update customer guidance on the Radware customer portal as new information becomes available.



EFFECTIVE DDoS PROTECTION ESSENTIALS

Hybrid DDoS Protection – Use on-premises and [cloud DDoS protection](#) for real-time [DDoS attack prevention](#) that also addresses high-volume attacks and protects from pipe saturation

Behavioral-Based Detection – Quickly and accurately identify and block anomalies while allowing legitimate traffic through

Real-Time Signature Creation – Promptly protect against unknown threats and zero-day attacks

Web DDoS Tsunami Protection – Automated immediate detection and mitigation of Web DDoS encrypted high RPS and morphing attacks

A Cybersecurity Emergency Response Plan – Turn to a dedicated emergency team of experts who have experience with Internet of Things security and handling IoT outbreaks

Intelligence on Active Threat Actors – High fidelity, correlated and analyzed data for preemptive protection against currently active known attackers

For further [network and application protection](#) measures, Radware urges companies to inspect and patch their network to defend against risks and threats.

EFFECTIVE WEB APPLICATION SECURITY ESSENTIALS

Full OWASP Top-10 coverage against defacements, injections, etc.

Low false positive rate using negative and positive security models for maximum accuracy

Auto-policy generation capabilities for the widest coverage with the lowest operational effort

Bot protection and device fingerprinting capabilities to overcome dynamic IP attacks and achieve improved bot detection and blocking

Securing APIs by filtering paths, understanding XML and JSON schemas for enforcement, and using activity tracking mechanisms to trace bots and guard internal resources

Flexible deployment options including on-premises, out-of-path, virtual or cloud-based

LEARN MORE AT RADWARE'S SECURITY RESEARCH CENTER

To know more about today's attack vector landscape, understand the business impact of cyberattacks, or learn more about emerging attack types and tools, visit Radware's [Security Research Center](#). Additionally, visit Radware's [Quarterly DDoS & Application Threat Analysis Center](#) for quarter-over-quarter analysis of DDoS and application attack activity based on data from Radware's cloud security services and threat intelligence.



THIS REPORT CONTAINS ONLY PUBLICLY AVAILABLE INFORMATION, WHICH IS PROVIDED FOR GENERAL INFORMATION PURPOSES ONLY. ALL INFORMATION IS PROVIDED “AS IS” WITHOUT ANY REPRESENTATION OR WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES THAT THIS REPORT IS ERROR-FREE OR ANY IMPLIED WARRANTIES REGARDING THE ACCURACY, VALIDITY, ADEQUACY, RELIABILITY, AVAILABILITY, COMPLETENESS, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT. USE OF THIS REPORT, IN WHOLE OR IN PART, IS AT USER’S SOLE RISK. RADWARE AND/OR ANYONE ON ITS BEHALF SPECIFICALLY DISCLAIMS ANY LIABILITY IN RELATION TO THIS REPORT, INCLUDING WITHOUT LIMITATION, FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, OR EXEMPLARY DAMAGES, LOSSES AND EXPENSES ARISING FROM OR IN ANY WAY RELATED TO THIS REPORT, HOWEVER CAUSED, AND WHETHER BASED ON CONTRACT, TORT (INCLUDING NEGLIGENCE) OR OTHER THEORY OF LIABILITY, EVEN IF IT WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, LOSSES OR EXPENSES. **CHARTS USED OR REPRODUCED SHOULD BE CREDITED TO RADWARE**

©2026 Radware Ltd. All rights reserved. The Radware products and solutions mentioned in this document are protected by trademarks, patents and pending patent applications of Radware in the U.S. and other countries. For more details please see: <https://www.radware.com/LegalNotice/>. All other trademarks and names are property of their respective owners.