



UPDATED THREAT ADVISORY

Update: Critical PHP2Shell Vulnerabilities



Services Performed By:

UltraViolet Cyber TIDE Team
tide@uvcyber.com

Update Published Date:

October 23, 2026

TLP:GREEN



Executive Snapshot

The UltraViolet Cyber Threat Intelligence and Detection Engineering (TIDE) team is publishing this Updated Threat Advisory in support of patch and remediation efforts to the PHP2Shell vulnerabilities, CVE-2026-109618 and CVE-2026-109619. Readers can find the original Emergency Threat Advisory [here](#).

PHP2Shell has drawn exploitation attempts both before and after public disclosure, but not all activity described as exploitation has been validated as a working attack. Amazon Threat Intelligence identified an attempt against its MadPot honeypot on October 17, four days before disclosure. Additional activity followed on October 21, when multiple exploit variants and requests from different anonymized infrastructure were observed. At the same time, some publicly circulated proof-of-concept (PoC) code was nonfunctional, and honeypot research found that much of the PHP2Shell-labelled traffic consisted of basic probes and checkers rather than exploit payloads. These distinctions matter: the available evidence confirms active targeting, but does not establish widespread successful compromise. Organizations should patch affected PHP runtimes immediately, apply the latest PHP configuration guidance, and investigate suspicious activity dating back to at least October 17.

- **Patch immediately.** Fixed releases are available for PHP 8.5, 8.4, 8.3, and 8.2. Patching is the only complete remediation for CVE-2026-109618 and CVE-2026-109619.
- **Apply the October 23 PHP configuration guidance.** PHP reiterated its recommendation to disable session upload progress until the next regular bugfix release. This setting was already recommended on October 21 for CVE-2026-109619.
- **Account for activity before disclosure.** Amazon recorded an attempt against MadPot on October 17. Further attempts were observed on October 21, with variations in payloads and anonymization infrastructure suggesting activity by multiple actors. The available reporting does not confirm successful compromise from these observations. UltraViolet Cyber is supporting this by performing extended threat hunts going back 90 days from disclosure.



Updated PHP Remediation Guidance

Patch Information

The PHP project released fixes for CVE-2026-109618 and CVE-2026-109619 on October 21, 2026.

PHP branch	Fixed release
PHP 8.5	8.5.12
PHP 8.4	8.4.27
PHP 8.3	8.3.36
PHP 8.2	8.2.35

Organizations should update the PHP runtime, including PHP-FPM, Apache mod_php, CGI, FrankenPHP, LiteSpeed, and other deployed PHP execution paths. Updating an application, framework, or content management system does not resolve a vulnerable PHP core runtime unless the underlying runtime is also updated.

For PHP 8.1, 8.0, and 7.4, the PHP project has published source-only security backports, but no new tagged releases or official binaries. Organizations that build PHP themselves can use these source branches; others should obtain a patched package from their operating system, distribution, or long-term support vendor. For PHP 7.3 and earlier, no corresponding PHP-project backport is available. If no supported fix is available, prioritize migration to a supported, patched release.

If an immediate upgrade is not possible, restrict public access or place the affected service behind additional controls where operationally feasible. Treat this as a temporary risk reduction measure, not remediation.

New PHP Configuration Guidance

PHP's October 23 informational advisory reiterates the recommendation to set: `session.upload_progress.enabled=0`

Keep this setting in place until the next regular PHP bugfix release. It was already listed as a mitigation for CVE-2026-109619 in the October 21 advisory. Organizations that applied it correctly should leave it in place and do not need to repeat the change.

The October 23 advisory also describes a separate issue, [CVE-2026-115560](#), involving an unusual combination of configuration and application behavior. The scenario requires the application to configure `session.serialize_handler` as `php_binary` or `php_serialize`, switch it to `php` at runtime using `ini_set()` before calling `session_start()`, and leave session upload progress enabled. PHP has indicated that it will publish a full advisory with a future release.

PHP states that applications that never change the session serialization handler at runtime can keep session upload progress enabled for this additional issue. However, its published recommendation is to disable the feature until the next regular bugfix release. This is separate from patching the original PHP2Shell vulnerabilities.

Implementation requirements:

- Set the directive in `php.ini`, an appropriate PHP-FPM pool configuration, or another server-level configuration that takes effect before request parsing.



- Restart or reload the relevant PHP services so that every worker uses the updated setting.
- Do not rely on `ini_set()` or PHP-FPM `.user.ini` for this mitigation. They do not provide the required protection before request parsing.
- Verify the effective setting in the running environment rather than assuming that editing a configuration file has updated every worker.

Additional Mitigations

Please note that these settings reduce exposure to specific failure conditions. They do not eliminate the underlying vulnerability, and they should not be used as a reason to defer patching. The best proactive measure remains immediate patching of the PHP2Shell vulnerabilities.

CVE-2026-109619 involves stale internal state that can persist after a request-owned memory region has been freed. A subsequent request handled by the same worker may encounter that state before normal initialization resets it.

- **PHP 8.5:** Set `fatal_error_backtraces=0` and restart every PHP worker. This addresses the confirmed fatal-error backtrace path only.
- **Remove problematic per-request configuration.** Review invalid or deprecated PHP settings applied during request processing, particularly those that can emit diagnostics before initialization is complete. Suppressing messages with `error_reporting`, `display_errors`, or the `@` operator does not prevent the underlying condition.
- **Address attacker-reachable fatal errors and timeouts.** Review `upload`, `import`, `callback`, `error-handling`, and `cleanup` paths that can be triggered by untrusted requests.
- **Consider worker recycling.** Where performance and application behavior permit, setting PHP-FPM `pm.max_requests=1` limits each worker to one request before recycling. This may disrupt attack paths that depend on state persisting between requests, but it is resource-intensive and should be tested before deployment.

CVE-2026-109618 involves memory corruption associated with out-of-memory handling. PHP recommends additional controls for deployments that cannot immediately be patched.

- **Remove PHP's controllable memory-limit failure.** Set `memory_limit=-1` in a configuration applications cannot override, such as an appropriate PHP-FPM administrative setting. Verify that other pool, directory, or application settings do not reintroduce a finite limit. This removes one controllable failure condition, but an underlying allocation failure can still reach the vulnerable path.
- **Limit request-body size.** PHP recommends `post_max_size=8M` for applications that do not require file uploads and `post_max_size=32M` where uploads are required. Keep `upload_max_filesize` below `post_max_size`. Validate these values against application requirements before deployment.
- **Enforce request limits upstream.** Where supported, apply limits at the web server, reverse proxy, or gateway for decoded request size, input field count, multipart part count, header count, and nesting depth. Ensure the controls also cover requests without a usable Content-Length header.
- **Use resource limits as an availability boundary.** Container or cgroup limits can constrain resource consumption. Where possible, configure the environment to isolate or terminate the affected worker rather



than the entire service. Do not substitute `RLIMIT_AS` or `RLIMIT_DATA` for these controls, as those limits can trigger the vulnerable allocation-failure path.

Additional Network and WAF Controls During Patching

Web application firewall (WAF) and reverse-proxy controls can reduce exposure while patching is underway. Their effectiveness depends on the application's traffic patterns and the attack path being addressed.

- If the application does not require file uploads, consider blocking multipart/form-data requests at the edge.
- Where uploads are required, apply appropriately scoped request-body limits and rate controls without disrupting legitimate traffic.
- AWS WAF's PHP application managed rule group version 2.3 includes the `PHP2Shell_BODY` rule for CVE-2026-109619. This rule does not address CVE-2026-109618's out-of-memory corruption path.
- Request-body limits can reduce exposure to oversized requests, but may not catch all memory-amplification scenarios, including multipart headers that consume disproportionate server-side memory.

Test WAF and proxy changes in monitoring or count mode where possible, review legitimate traffic matches, and then enforce the rules. A blocked test request or a clean WAF dashboard does not establish that the PHP runtime is fully protected.

TIDE Team Analysis

Attempted Exploitation

Amazon Threat Intelligence identified an attempt against its MadPot honeypot on October 17, 2026, four days before the PHP2Shell vulnerabilities were publicly disclosed. The attempt originated from an IP address associated with a commercial anonymization network. Amazon stated that the targeting of a honeypot from anonymized infrastructure suggested threat actor activity, but it could not determine how the actor obtained exploit information before disclosure.

On October 21, Amazon identified multiple PHP2Shell PoC variants, with the earliest observed around 02:00 UTC. Additional attempts against MadPot between approximately 02:00 and 08:00 UTC came from other anonymized infrastructure. Variations in payloads and anonymization choices suggested that multiple actors were involved. Amazon's technical analysis did not establish whether these attempts would have succeeded against real-world systems.



After public disclosure and patch release, additional PoCs appeared within hours and were used against Amazon sensors. Amazon described PHP2Shell as being broadly exploited in the wild. This supports the conclusion that exploitation attempts expanded quickly, but it should not be read as confirmation that all observed requests were functional exploits or that widespread successful compromise had been established.

Public POCs and Scanning Activity

The public PoC landscape was inconsistent during the initial disclosure period. The PHP security team stated that PoCs published before the patches were released were not functional. Subsequent reporting also identified public repositories and scripts whose claims were not supported by the code, whose checks did not demonstrate remote code execution, or whose behavior depended on specific lab configurations.

This distinction matters for threat intelligence and incident response. A repository labelled as a PoC is not necessarily a working exploit. Likewise, a request containing a PHP2Shell user-agent string, a probe for `PHP_SESSION_UPLOAD_PROGRESS`, or a basic checker request does not establish that the sender had a functional exploit.

Honeypot observations published after disclosure also found that much PHP2Shell-labelled traffic consisted of simple checks, parser fuzzing, and generic scanning. One set of observations identified crafted serialized payloads consistent with an exploitation attempt after disclosure, while many other requests did not carry exploit-like content. This reinforces the need to classify activity according to the evidence available, rather than treating all traffic associated with the vulnerability as successful exploitation.

At the same time, nonfunctional public PoCs do not negate the independent evidence of exploitation attempts reported by Amazon and other researchers. Nor do they establish that no working exploit existed privately. The defensible assessment is that the vulnerabilities were actively targeted, exploit development was progressing rapidly, and the success rate of the observed activity remains unclear.

Mirai-Related Activity

Security researchers have reported PHP2Shell exploitation attempts associated with efforts to deploy Potassium, a Mirai-derived botnet. The reporting reviewed does not confirm that these attempts resulted in successful PHP2Shell-

VULNERABILITY PROFILE

CVE-2026-109618

Unauthenticated RCE via Out-of-Memory Corruption in PHP Core (PHP2Shell)

CRITICAL v4.0 **9.2**

FIRST EXPLOITED

Attempted exploitation confirmed, Amazon's MadPot honeypot logged an attempt on October 17, four days before public disclosure. Success against real-world systems was not established. **KEV** Not listed.

SOFTWARE

PHP Core by The PHP Foundation

AFFECTED

PHP 8.2 before 8.2.35, PHP 8.3 before 8.3.36, PHP 8.4 before 8.4.27, and PHP 8.5 before 8.5.12.

PATCH

8.5.12 / 8.4.27 / 8.3.36 / 8.2.35 Released October 21, 2026, with CVE-2026-109619. Update the runtime itself, not the application layer. [GHSA-w8hw-w9fx-j79w](#)

VULNERABILITY PROFILE

CVE-2026-109619

Unauthenticated RCE via State Global State in PHP Core (PHP2Shell)

CRITICAL v4.0 **9.2**

FIRST EXPLOITED

Attempted exploitation confirmed, Amazon's MadPot honeypot logged an attempt on October 17, four days before public disclosure. Success against real-world systems was not established. **KEV** Not listed.

SOFTWARE

PHP Core by The PHP Foundation

AFFECTED

PHP 8.2 before 8.2.35, PHP 8.3 before 8.3.36, PHP 8.4 before 8.4.27, and PHP 8.5 before 8.5.12.

PATCH

8.5.12 / 8.4.27 / 8.3.36 / 8.2.35 Released October 21, 2026, with CVE-2026-109618. Update the runtime itself, not the application layer. [GHSA-6wmf-4rv5-p3rp](#)



enabled infections.

This activity illustrates the potential for vulnerable PHP systems to be targeted for botnet recruitment as well as direct access. Defenders should monitor for unexpected child processes, shell execution, new persistence mechanisms, and unexplained outbound connections from PHP workers or the hosts running them.

Exploit Development

Security researchers have reported developing an exploit for CVE-2026-109618 that uses a memory leak to defeat Address Space Layout Randomization (ASLR) before achieving remote code execution on x86-64 systems, including configurations with PHP error display disabled. The researchers have not publicly released the exploit.

This development is significant because it demonstrates that ASLR and disabled error display should not be treated as sufficient barriers to exploitation. However, this capability is privately held rather than public, and a research demonstration against a particular environment is not evidence that every vulnerable PHP installation is exploitable under all configurations.

The broader lesson is that defenders should not rely on individual hardening settings to compensate for an unpatched runtime. Confirmed exploit demonstrations, public PoC claims, and observed attack traffic should remain separate categories in reporting and investigation.

Implications for Defenders

PHP2Shell affects the PHP runtime rather than a single application or framework. Any externally reachable service using a vulnerable runtime should be considered in scope, including systems that are not directly managed by application teams, such as vendor appliances, control panels, and embedded web interfaces.

The primary concern remains unauthenticated remote code execution. Successful exploitation could provide an attacker with a foothold for follow-on activity, including credential access, deployment of malware or botnet components, lateral movement, and data theft, depending on the privileges and network access of the affected service.

The activity reported before disclosure means that investigations should not begin solely on October 21. However, defenders should also avoid overstating the evidence by counting every PHP2Shell-labelled request or public PoC as a confirmed exploit. Focus on the quality of the observed request, the affected runtime and configuration, the outcome of the request, and any corroborating host activity.

Patching remains the priority. Configuration mitigations and edge controls can reduce exposure while updates are deployed, but neither a lack of alerts nor the absence of a known PoC match proves that a vulnerable system was not targeted.

Why It Matters

The PHP2Shell activity highlights how quickly a critical vulnerability can attract attention, even before public disclosure. Amazon Threat Intelligence observed an attempt against its MadPot honeypot on October 17, four days before disclosure, followed by additional attempts on October 21. While some publicly circulated PoCs were nonfunctional and much of the observed traffic consisted of basic probes rather than demonstrated exploitation, the activity shows that PHP2Shell was being actively investigated and targeted as technical details emerged. The



distinction is important: not every request represents a working exploit, and not every exploitation attempt results in compromise. However, organizations should not interpret ineffective public PoCs as evidence that the underlying vulnerabilities pose little risk, particularly as exploit development continues.

This pattern is consistent with previous high-profile vulnerabilities such as Log4Shell, where public disclosure accelerated scanning and exploit development across a broad range of exposed systems. The circumstances are different, but the operational challenge is similar. Once technical details become available, attackers can adapt existing research, test vulnerable systems, and develop new approaches while defenders work to identify affected assets and deploy fixes. PHP's widespread use across web applications, hosting platforms, control panels, and embedded products adds another challenge: vulnerable runtimes may exist outside an organization's primary application inventory or be managed by third-party providers. Teams must account for those dependencies rather than assuming that updating the application itself is sufficient.

For defenders, the priority is to patch affected PHP runtimes, apply PHP's recommended configuration mitigations, and investigate suspicious activity dating back to October 17. The available reporting confirms exploitation attempts, but does not establish widespread successful compromise. Investigations should distinguish routine scanning and ineffective PoCs from requests that contain credible exploit activity, then correlate suspicious requests with endpoint telemetry and evidence of post-exploitation behavior. Patching also does not rule out an earlier compromise, so systems with suspicious activity may require further investigation even after the vulnerable runtime has been updated. The key takeaway is to respond quickly without overstating the evidence: reduce exposure, verify remediation, and use observed activity to guide targeted investigation.



How to Respond

- **Patch every affected PHP runtime.**
 - Inventory PHP versions across internet-facing and internal systems, including PHP-FPM, mod_php, CGI, FrankenPHP, LiteSpeed, appliances, and control panels.
 - Upgrade supported branches to PHP 8.5.12, 8.4.27, 8.3.36, or 8.2.35, as applicable.
 - For older or unsupported branches, verify whether the distribution or long-term support vendor has provided a backported fix. Otherwise, plan migration to a supported release.
 - If patching cannot be completed immediately, restrict public access or place the service behind additional controls where operationally feasible.
- **Apply and verify the PHP configuration mitigations.**
 - Set `session.upload_progress.enabled=0` in server-level configuration and restart or reload every relevant PHP worker. Keep it disabled until the next regular PHP bugfix release.
 - On PHP 8.5, set `fatal_error_backtraces=0` and restart every relevant worker.
 - Review per-request INI configuration for invalid or deprecated settings that may generate diagnostics before initialization.
 - For CVE-2026-109618, apply the recommended `memory_limit`, `request-size`, `upload-size`, and resource-boundary controls where compatible with the application.
 - Verify effective settings after service restarts. A configuration file edit alone is not proof that every worker is using the intended values.
 - Review whether applications change the session serialization handler at runtime.
- **Verify patch coverage before standing down.**
 - Confirm the installed runtime version using the package manager, vendor advisory, package changelog, or runtime binary, not the application version alone.
 - Check each PHP execution path and service instance, including systems maintained by third parties.
 - Confirm that all workers have restarted and that the expected configuration is effective.
 - Track remaining exceptions to completion and prioritize internet-facing systems first.
- **Apply compensating network controls to systems awaiting remediation.**
 - Block multipart uploads where the application does not require them.
 - Where uploads are necessary, use scoped request-body limits, rate controls, and other appropriate edge protections.



- Confirm the limits do not interfere with legitimate application traffic.
- Treat AWS WAF's PHP2Shell_BODY rule as a mitigation for CVE-2026-109619 only. It does not fully address CVE-2026-109618.
- Document any service that remains exposed until patching is complete, along with its temporary controls and remediation deadline.
- **Investigate for exploitation attempts and potential compromise.**
 - Review web server, reverse proxy, WAF, PHP-FPM, application, and endpoint telemetry from October 17, 2026, onward.
 - Look for suspicious multipart POST requests, unexpected PHP worker child processes, shell execution, unusual outbound connections, and unexpected file or account changes.
 - Review relevant telemetry for fatal errors, timeouts, and repeated request patterns that may indicate exploitation attempts.
 - Distinguish exploit attempts from generic scanning and nonfunctional PoC activity. A PHP2Shell user-agent string or a request matching a public script is an investigative lead, not proof of exploitation.
 - Use published indicators as investigative leads, not as a complete blocklist. Some observed infrastructure is associated with anonymization services, and absence of a match does not establish that a system was not targeted.
 - If evidence of successful exploitation is found, follow the organization's incident response process. Isolate affected systems as appropriate, preserve logs and forensic evidence, investigate persistence and lateral movement, and rotate credentials or secrets that may have been exposed.
- **Maintain monitoring until remediation and investigation are complete.**
 - Monitor for continued exploitation attempts and unusual activity from PHP runtimes.
 - Recheck systems that were offline, missed during the initial inventory, or dependent on vendor-managed updates.
 - Keep temporary controls in place until fixed versions are verified and any investigation is complete.
 - Do not use the absence of alerts, a successful WAF test, or the presence of a single configuration mitigation as evidence that an installation is safe.

What UltraViolet Cyber is Doing

- Proactively enabling custom detections based on the intelligence available for the PHP2Shell vulnerabilities. UltraViolet is also maximizing detection coverage proactively through [Equinox](#).
- Performing hypothesis driven threat hunts based on the intelligence available for the PHP2Shell vulnerabilities. UVCyber customers will be informed of the results through secure channels.



- Parsing available victim dump data for any social, financial, business, or technical relations to UVCyber Clients and partner organizations.
- Aggregating threat intelligence from myriad sources and applying the most up-to-date knowledge to proactive threat hunting and response.



About UltraViolet Cyber

UltraViolet Cyber is a leading tech-enabled managed security services provider, delivering unparalleled cybersecurity expertise that fills technology and talent gaps across Global 2000 and Federal Government customers. Founded and operated by security practitioners from the national intelligence community, UltraViolet Cyber connects offensive security, application security, detection and response, and security engineering to deliver a differentiated approach to cybersecurity operations. Transforming customers' security programs, UltraViolet Cyber's flagship security-as-a-service solution, UV Lens, removes complex operational silos, replacing them with integrated security capabilities. UltraViolet is headquartered in McLean, Virginia with technology centers across the world.

443.351.7630 / info@uvcyber.com |  UltraViolet Cyber |   @uv_cyber
