



PHP2Shell: Monitoring for related compromises

by

Cyber Security Agency of Singapore (CSA)

VERSION 1.0

23 OCTOBER 2026



Preface

First reported publicly on 20 October 2026, PHP2Shell refers to two critical vulnerabilities (CVE-2026-109618 and CVE-2026-109619) in the PHP runtime that runs on web servers and serves many websites and web applications. These vulnerabilities may allow unauthenticated attackers to execute code on vulnerable servers. The issue is in PHP itself, not just in any one website product. [1-3]

Since then, PHP had issued patched versions on 21 October 2026, and multiple reports of exploits have emerged. Patching PHP runtimes is a must and should be done pronto. Where affected systems cannot be patched immediately, organisations should consider (1) temporarily taking them offline or restricting internet access or (2) implement the temporary mitigation measures recommended by the PHP security team until the relevant security updates can be applied, considering operational requirements.

Where these are not feasible, organisations should **use this guide to monitor for potential exploitation attempts**, including:

- Unusual or repeated HTTP POST requests, particularly those involving large or malformed request bodies.
- Unexpected PHP memory exhaustion errors, worker crashes or repeated HTTP 5xx responses.
- Suspicious execution of system commands or unexpected processes originating from PHP or web server processes.

Part A is designed for *website and service owners* to determine if they are using vulnerable builds of PHP runtime.

Part B is designed for *system administrators or security teams* to implement measures to monitor for PHP2Shell-related compromises.

The indicators laid out in this guide (specifically in Step 5) do not necessarily confirm successful exploitation and should be assessed in context. Organisations that identify signs of possible compromise should investigate promptly and follow their incident response procedures.

Important: Temporary mitigations and monitoring do not replace patching. Applying the relevant security updates remains the only complete mitigation for these vulnerabilities. If your service was unpatched for more than 12 hours since the release of the security update, ask the technical team responsible or provider to carry out the relevant checks in Step 5. **Do not delay patching while reviewing logs.**



PART A | For website owners and service owners

1. Does your website or application use PHP?

You may be using PHP even if your website address does not end with ".php". Start with the product you use, then confirm with the person or provider responsible for the server.

If your situation is...	What you should do now
WordPress, Joomla, Drupal or similar CMS	These products run on PHP. Confirm the PHP version and patch status with your site administrator or hosting provider. A CMS/plugin update alone does not patch PHP.
Website built by an external vendor	Ask whether PHP serves any website, API or administration function, including components bundled inside other software.
Self-managed website or application	Ask your administrator to verify the running web PHP build (not just a command-line installation) and follow Part B.
Unsure which technology is used	Contact the website developer or hosting provider. Treat PHP exposure as unresolved until they confirm it.
Confirmed no PHP is used	Record that conclusion in your normal asset records. No PHP2Shell-specific checks are needed for that service.

PHP is likely when you use a recognised PHP-based platform (for example WordPress, Joomla, Drupal, Magento/Adobe Commerce, Moodle, Nextcloud or MediaWiki). Other clues include URLs containing ".php" or "/wp-admin/", a "PHPSESSID" browser cookie, or an "X-Powered-By: PHP" response header. These clues can be absent, stripped or misleading; confirm with the application or hosting team before concluding that PHP is not used.

2. Find your PHP version and who manages it

WordPress: Tools > Site Health > Info > Server. Joomla: System > System Information. Drupal: Reports > Status report. These administration pages can show the PHP version in use. [7-9]

If a provider manages PHP, ask them: (1) Does the service use PHP and which build is running? (2) When was the fixed build deployed and were workers restarted? (3) Were relevant WAF alerts, PHP errors and signs of compromise checked during the exposure period? Updating a CMS or plugin alone does not fix PHP.



PART B | For administrators and security teams

Use Part B if you operate the PHP runtime, manage the application security monitoring, or need to validate a provider response. Begin with applicability and patch status, then select checks based on your deployment and the logs actually available.

3. Confirm the running PHP build and patch status

PHP branch	First upstream fixed version
8.5	8.5.12 or later
8.4	8.4.27 or later
8.3	8.3.36 or later
8.2	8.2.35 or later
8.1 and older	Older branches need vendor/distribution-confirmed backports or migration to a supported patched release.

Verify the PHP interpreter serving HTTP requests and confirm that all old workers were restarted or containers replaced after patching. Linux distributions may backport fixes without changing the major version string. Check the security status of the installed package, not simply the output of "php -v". [1-3]

Prioritise exposed installations

Prioritise internet-facing PHP services, particularly Apache mod_php deployments. PHP-FPM, CGI, other SAPIs, containers and vendor-bundled PHP also require assessment; do not assume they are unaffected. If the service was vulnerable before patching, search available logs from 17 October 2026 until the patched build was operating, and extend as evidence warrants. [1-4,6]

For administrators: ways to establish PHP usage

Environment	Configuration or runtime evidence
Linux / Apache / Nginx	Identify running php-fpm, php-cgi, lsphp or FrankenPHP workers. For Apache, check loaded PHP modules and handlers; for Nginx, review fastcgi_pass and the actual PHP backend.



Environment	Configuration or runtime evidence
Containers / Kubernetes	Inspect the running deployment, PHP interpreter inside each relevant container and the deployed build, not just the image tag.
Windows / IIS	Check IIS Handler Mappings and FastCGI Settings for php-cgi.exe or another interpreter used by the application.



4. Choose the monitoring route for your deployment

How a service is hosted determines who can access its logs. Whether a WAF is deployed is a separate question: a WordPress site, for example, may run on a cloud VM with or without a WAF. Apply both the hosting guidance and the log-availability guidance.

Deployment	Who should check / Likely evidence
On-premises or self-managed cloud VM	Server/application team: web or load balancer access logs, PHP/Apache/PHP-FPM error logs, EDR/OS process events and WAF events if deployed.
Containers, Kubernetes or managed app platform	Platform/application team: ingress/WAF logs, application stdout/stderr, deployed PHP image and workload process telemetry. Capture ephemeral logs before workloads roll over.
CMS under shared hosting	Website owner and hosting provider: version/status via CMS/control panel, provider WAF/access/PHP logs; host process logs may require provider support.
Fully managed hosting or SaaS	Service provider: confirm PHP applicability and patches; provide summary of WAF matches, PHP errors and signs of compromise if the customer cannot access raw logs.

If you have a WAF or CDN security service

Start with the vendor's PHP2Shell-specific detections, the date rules became active and the rule action (blocked, challenged, counted/logged or allowed). For AWS WAF, review "PHP2Shell_BODY" in AWSManagedRulesPHPRuleSet v2.3+ for CVE-2026-109619; AWS also recommends request-body size controls such as "SizeRestrictions_BODY" for CVE-2026-109618. For Cloudflare, inspect the PHP RCE managed-rule events and the corresponding request/response context. Rules and coverage differ by vendor. [4-5]

Next, examine abnormal POST requests, correlate any allowed traffic with PHP error and process logs, and note request-body inspection limits. A WAF does not replace patching.

If you have web access logs but no WAF

Start with POST requests, URLs, request length where recorded, and 5xx responses. Then check PHP errors and process events. Standard Apache/Nginx access logs commonly expose response bytes rather than incoming request-body size; confirm the field before using a size threshold. Standard access logs normally cannot reveal long form field names or "PHP_SESSION_UPLOAD_PROGRESS" parameters.



If you only have PHP error logs or EDR

PHP logs: start with memory-exhaustion strings and worker crashes (Checks 2 and 5). EDR: start with unusual web/PHP parent-child process executions (Check 3), then review file and network evidence if suspicious. If logging is absent, record the visibility limitation internally and ensure the runtime is remediated; no matching logs is not the same as a clean finding.

Special note for WordPress, Joomla, Drupal and other CMS

Media uploads and plugins legitimately generate large multipart POSTs. Compare requests with normal upload endpoints and expected usage. Unexpected large POSTs to pages that normally do not accept uploads are more interesting, but are not proof of exploitation. Some plugins legitimately invoke utilities such as zip, convert or mysqldump; investigate unexpected command lines and accounts against a known-good baseline.

5. Run the prioritised monitoring checks

The filters below are logical search conditions for use in WAF, SIEM, web/PHP log viewers or EDR. They are not literal vendor query syntax. Run P1 on potentially vulnerable deployments wherever the logs exist; use P2 to corroborate; extend to P3 where findings remain unexplained.

P1 - Essential checks

Check / log source	What to search for	What a match means / next check
1. Large or suspicious POST WAF / proxy / access logs	Method = POST AND Content-Type contains either "application/x-www-form-urlencoded" or "multipart/form-data". Review unusually large request bodies/lengths (start near 1 MB if this field exists), anomalous routes, extreme form field lengths or PHP2Shell WAF matches.	Review source, URI, time, size and WAF action. Check whether traffic reached PHP. Compare with legitimate form/upload traffic and correlate with Checks 2 and 3.
2. PHP memory exhaustion PHP / Apache / PHP-FPM logs	Message contains BOTH "Allowed memory size" AND "exhausted"; OR contains "Out of memory". Prioritise errors also containing "Unknown on line 0".	Identify host and time. Review relevant POST requests just before the error (start around 10 seconds, widen if needed). PHP memory-limit exhaustion is distinct from an OS OOM-killer event.



Check / log source	What to search for	What a match means / next check
3. Unexpected shell or utility execution EDR / process audit logs	Parent: apache2, httpd, php, php-fpm*, php-cgi, lsphp or FrankenPHP. Child: sh, bash, dash, curl, wget, python*, perl or other unusual utilities. On IIS, include php-cgi.exe/w3wp.exe to cmd.exe/powershell.exe.	Inspect the full process tree, command line, account and preceding HTTP requests. Unauthorised execution is a strong possible compromise indicator: follow established incident response procedures promptly.

The ~1 MB threshold is an initial screening choice, not a validated PHP2Shell signature. Content-Type alone is normal web traffic. Datadog demonstrated a very long URL-encoded form field and documented shell execution without requiring an HTTP error or worker crash. [3]

P2 - Corroborating checks

Check / log source	What to search for	How to use the result
4. Session upload progress WAF with request inspection	POST where inspected body contains "PHP_SESSION_UPLOAD_PROGRESS" and a PHPSESSID cookie or matching body value.	Confirm whether the application legitimately uses PHP upload progress. Look for related requests, PHP errors and possible same-worker sequences if PID information exists. PHPSESSID alone is normal.
5. Worker crashes / HTTP 5xx PHP/Apache/OS and web logs	Service errors: "Segmentation fault", "SIGSEGV", "AH00052", worker exits. Separately HTTP Method = POST AND Status IN (500, 502, 503); identify bursts.	Check for suspicious requests before errors, and for repeated faults on a worker. Neither a single 500 response nor a crash confirms exploitation; CVE-2026-109619 can span multiple requests.



P3 - Deeper checks when warranted

What to inspect	Reason and follow-up
Unexpected outbound connections	Identify unusual destinations contacted by web/PHP processes, especially shortly after suspicious HTTP activity or unauthorised command execution.
New or changed files	Review new PHP scripts, executables and altered application files in web roots or temporary paths such as /tmp and /var/tmp; compare with authorised deployment changes.
Unusual PHP diagnostics and request sequences	Review PHP fatal/parse errors, warnings, timeouts and repeated POSTs only when associated with stronger suspicious signals. Same-worker correlation may help for CVE-2026-109619.
Possible data access or spread	If compromise is suspected, review access to secrets and databases and unexpected internal connections from the PHP host.

6. How to interpret the checks

Observation	Practical interpretation
Only a large POST or 5xx response	A weak anomaly; check the application baseline, patch status and whether other suspicious events exist.
Suspicious POST plus PHP memory error or crash	Possible exploit attempt; determine WAF outcome and correlate with process/network evidence.
Unauthorised PHP/web-worker shell or malicious file	Possible compromise; preserve relevant evidence and use normal incident response procedures.
No suspicious matches or insufficient logs	Not proof of safety. Successful exploitation can return HTTP 200 and remain inside the worker; absent request-body fields cannot be searched retrospectively.

Do not enable unrestricted HTTP request/response-body logging solely for this exercise: request bodies can contain passwords and personal data. Prefer existing metadata, managed WAF alerts and targeted authorised inspection. For systems patched after exposure began, still check the period before the patched workers were running. [3-6]



References and further reading

- [1] [PHP project - CVE-2026-109618 \(OOM corruption; fixed versions and mitigation\)](#)
- [2] [PHP project - CVE-2026-109619 \(stale global state; multi-request behaviour\)](#)
- [3] [Datadog Security Labs - PHP2Shell detection and analysis \(21 October 2026\)](#)
- [4] [AWS Security Blog - PHP2Shell WAF mitigations \(21-22 October 2026\)](#)
- [5] [Cloudflare - PHP2Shell WAF protections \(21 October 2026\)](#)
- [6] [AWS Threat Intelligence - exploitation observed from 17 October 2026](#)
- [7] [WordPress - Site Health screen](#)
- [8] [Joomla - System Information documentation](#)
- [9] [Drupal - Site status report](#)