

SECURITY ADVISORY • EBOOK

PHP2Shell

The complete technical guide to the two critical PHP runtime vulnerabilities sitting beneath roughly **70% of the internet** — and exactly what defenders do about them.

CVE-2026-109618

CVE-2026-109619

22 yrs

BUG CLASS AGE

5.4.0+

PHP AFFECTED

Pre-auth

NO CREDENTIALS



Executive summary

PHP2Shell is a problem you have whether or not your own applications use PHP. Two critical vulnerabilities in the PHP runtime were patched on **October 21, 2026**, and they reach back to any install on PHP 5.4.0 or later — roughly 70% of the internet, which means your SaaS vendors, hosts, bundled appliances and integrations too.



CVE-2026-109618

Unauthenticated remote code execution in the PHP runtime, under default configs.



CVE-2026-109619

A second pre-auth RCE flaw that can also expose memory through stale references in reused PHP workers.



The runtime, not the app

Patching WordPress, Drupal or a dependency fixes neither. Only a patched PHP binary does.

Three realities frame the response

REALITY 1 • ATTACKERS MOVED FIRST

Amazon's MadPot sensor network observed PHP2Shell exploitation attempts **before** the public advisory. The window between first attacker activity and your patch is where the real incident risk now lives — and you can't assume you were untouched during it.

REALITY 2 • POCS AND SCANNERS ARE PUBLIC

Proof-of-concept exploits and free scanners (Astra's included) are in wide circulation. The same signatures that help defenders check exposure let threat actors map vulnerable targets at scale. Time-to-exploitation is now measured in **hours, not weeks**.

REALITY 3 • THE AGENCIES WERE UNUSUALLY DIRECT

CISA required immediate patching; NSA Cyber recommended taking affected systems **offline until fully mitigated** — language it does not use lightly. National CERTs across the UK, EU, Canada, Australia, Japan, India and more followed within hours.

WHAT THIS MEANS OPERATIONALLY

Patch every PHP runtime in your estate on supported branches (8.2.35 · 8.3.36 · 8.4.27 · 8.5.12). Backport or isolate end-of-life branches. Verify every patch after process restart. Rotate credentials used during the pre-disclosure window. And run a compliance review in parallel — this shape of incident triggers GDPR, SEC 8-K, DORA, NIS2, HIPAA, PCI-DSS, SOC 2 and ISO 27001 obligations at once.

NAVIGATE BY ROLE

What's inside

Executives: read the summary opposite and Chapter 8. Engineering leads: Chapters 2 and 6. Incident response: Chapters 5, 7 and 10. Board members: the briefing template in Chapter 12.

01	No one is safe from PHP2Shell	EVERYONE	04
02	Anatomy of the vulnerabilities	ENGINEERING	06
03	Who is actually affected	ENGINEERING	09
04	Detection: what works, what doesn't	SECURITY	11
05	Exploitation in the wild	INCIDENT RESPONSE	13
06	Remediation done properly	ENGINEERING	15
07	Post-exposure: assume you were a target	INCIDENT RESPONSE	17
08	Compliance and regulatory obligations	LEGAL · EXEC	19
09	The supply-chain dimension	VENDOR RISK	21
10	Response playbook for the first 72 hours	INCIDENT RESPONSE	23
11	How Astra Security helps	SECURITY	24
12	Checklist & board briefing reference	EXEC · BOARD	25

"In 2004 a researcher found a bug in PHP's memory handling worrying enough to keep the exploit to himself. Twenty-two years later, that same class of bug came back as PHP2Shell — and this time it produced the catastrophe."

— PREFACE

01

CHAPTER ONE

No one is safe from PHP2Shell

TL;DR

The vulnerability lives in the runtime beneath ~70% of the web — so "do you run PHP?" is the wrong question. Somebody in your supply chain almost certainly does, which makes this a supply-chain problem first and an engineering problem second.

The question is not "do you run PHP"

If you run PHP, you already know you have a problem. If you don't, you probably still do — because somebody in your supply chain almost certainly does. PHP runs roughly 70% of the websites on the internet, and for PHP2Shell that number stops being background noise and starts being the whole story.

When a vulnerability sits in the runtime beneath 70% of the web, the question of **who is affected stops being technical and starts being economic.**

Your SaaS vendor probably runs PHP somewhere. Your host definitely does. The third-party support tool you installed last year ships with a PHP container. The CRM you integrated uses a PHP admin panel. The vulnerability lives in the runtime itself — so every one of those systems inherits the risk **even if the application code on top of it is clean.**

The two CVEs

PHP2Shell refers to two vulnerabilities disclosed together on October 21, 2026 and patched in the PHP October 2026 Security Release.

CRITICAL

CVSS 9.2

CVE-2026-109618 • RCE

An unauthenticated remote code execution path in the PHP runtime, exploitable under default configurations, affecting versions as far back as 5.4.0.

CRITICAL

CVSS 9.2

CVE-2026-109619 • Pre-auth RCE

A PHP use-after-free flaw that can expose memory or enable unauthenticated code execution when a subsequent request accesses stale references in the same worker.

Both were discovered by Calif Research, OpenAI (as part of its Patch the Planet initiative) and Wiz, working with the PHP Foundation. Both can enable pre-auth RCE; CVE-2026-109619 can also expose memory containing credentials, tokens and API keys.

The agencies showed up at the same time

Within hours of the advisory, CISA directed immediate patching. NSA Cyber went further — telling organizations to **take affected systems offline until they can be fully mitigated**. That post reached 11.3 million views, a signal of how far outside the normal security-industry audience this disclosure travelled. The Wall Street Journal and New York Times both covered it.

THE PRE-DISCLOSURE GAP

Patches landed October 21. Amazon's MadPot network saw probing **before** October 21. The window between first known attacker activity and the first public advisory is where the real incident-response work will live for the next few quarters — because systems that are now patched may already have been touched, and their owners don't necessarily know yet.

02

CHAPTER TWO

Anatomy of the vulnerabilities

TL;DR

Both CVEs involve unsafe PHP memory handling and can enable pre-auth RCE. One involves out-of-memory corruption; the other retains stale references to freed request memory, which can also expose sensitive data.

The core defect, in one sentence

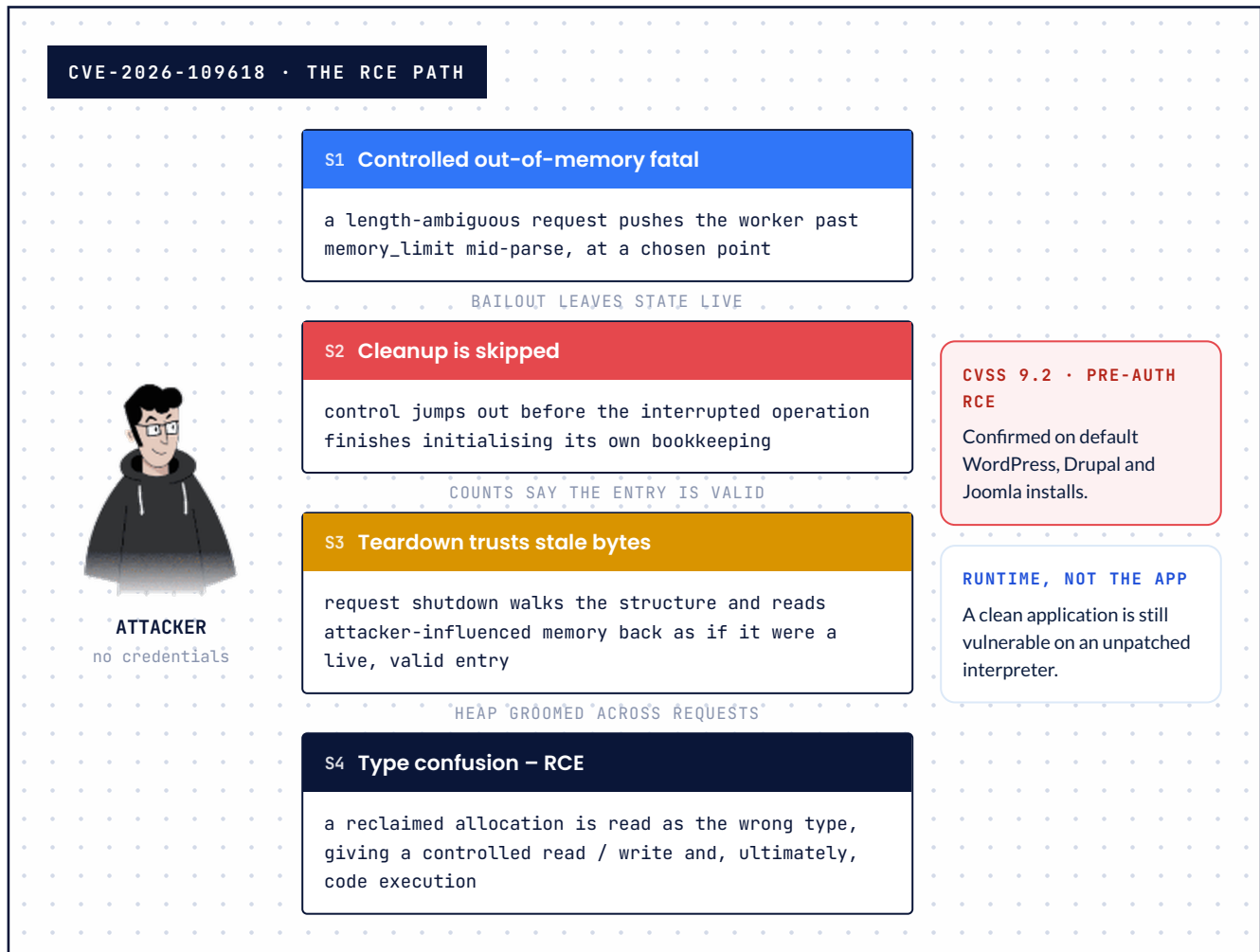
PHP2SHELL IN A SENTENCE

PHP can leave **dangling pointers and partially-initialised state** when a memory-limit fatal interrupts native code before cleanup completes — and request shutdown then frees memory that surviving state still references, giving an attacker a path to arbitrary code execution and sensitive memory contents.

Everything else is detail on how to get the interpreter into that state. The sections below walk the two exploitation paths at the level a defender needs to brief an engineering team — not a reproduction recipe.

Two paths to unsafe memory access

CVE-2026-109618 involves allocation-failure corruption; a confirmed path triggers PHP's configured memory limit. CVE-2026-109619 involves stale global references to freed request memory. A subsequent request in the same worker can access those references, exposing memory or enabling code execution. Each CVE requires its own temporary mitigations.



How CVE-2026-109618 turns a controlled fatal error into use-after-free and type confusion.

Server APIs and deployment modes

PHP deployments extend beyond Apache `mod_php` and PHP-FPM to LiteSpeed/LSAPI, `php-cgi`, FrankenPHP and CLI workloads. Startup, shutdown and worker reuse differ, so exploitation paths vary by deployment. The examples below describe `mod_php` and PHP-FPM; they are not an exhaustive inventory. Identify every PHP runtime, check its patch status and restart the relevant processes after updating.

Apache `mod_php`

Shutdown reaches the stale structure directly. The originally-confirmed RCE path.

PHP-FPM

Reached via the generic unserializer after request startup succeeds — application-dependent, but real.

CVE-2026-109619: stale global state and pre-auth RCE

This use-after-free flaw can expose memory or enable unauthenticated code execution when a subsequent request accesses freed request memory through stale references in the same worker. Attackers may gain the worker's privileges or access secrets it handled, including database credentials, API tokens, session data and cloud keys.

WHY WORKER REUSE MATTERS

Stale global references can survive after request memory is freed. A subsequent request handled by the same worker may access that invalid memory, enabling disclosure or remote code execution.

Confirmed vulnerable applications

Pre-authentication RCE confirmed against default installations of a near top-ten of "PHP apps a typical enterprise has installed somewhere":

WordPress 7.1.2

Drupal 11.4.3

Joomla 6.1.4

MediaWiki 1.46.2

Roundcube 1.7.4

Magento 2.4.9

phpMyAdmin 5.2.3

Nextcloud 35.0.1

Laravel 13.11

Symfony 8.1

WHMCS

WHY THE APPLICATION LAYER CANNOT SAVE YOU

A WordPress update does not fix PHP2Shell. A Drupal update does not. A composer dependency bump does not. The vulnerability lives one layer below the application — patching the app gives you a patched app on a vulnerable interpreter, which still falls to the same exploit. **The only fix is to update the PHP runtime itself and restart the processes using it.**

03

CHAPTER THREE

Who is actually affected

TL;DR

Any PHP from 5.4.0 (March 2012) onward should be assumed affected. The hard part isn't the version range — it's finding where PHP is actually running inside your estate.

The version range is wider than most realise

The earliest exploitation path documented reaches back to **PHP 5.4.0**, released March 2012. Any install in the 5.4.0 – 8.5.11 range should be assumed affected until proven otherwise. The fixed versions in the October 2026 release are the four actively-supported branches:

Branch	Status	Fixed in
PHP 8.5.x	VULNERABLE < 8.5.12	8.5.12
PHP 8.4.x	VULNERABLE < 8.4.27	8.4.27
PHP 8.3.x	VULNERABLE < 8.3.36	8.3.36
PHP 8.2.x	VULNERABLE < 8.2.35	8.2.35
PHP 8.1 & earlier (to 5.4.0)	END-OF-LIFE	Backport / isolate / upgrade

THE END-OF-LIFE TRAP

Much of the internet runs PHP 7.4, with pockets of 7.3 or 7.2 inside legacy apps. These branches are EOL — the PHP Foundation ships no updates for them. But Linux distributions (RHEL, Ubuntu LTS, Debian) backport fixes into their packages. A `php -v` of `7.4.33` is only part of the story: the real question is whether **your distribution has released a backport**. Trust the package manager's security advisory, not the version string.

The hidden-exposure problem

The hardest part of "who is affected" is figuring out where PHP is actually running. Teams who think they don't run PHP usually find it in at least some of these places:

Hosting & admin panels

cPanel / Plesk bundle their own PHP interpreters for user sites, separate from the application.

Security product consoles

Management consoles are often shipped as self-hosted PHP web applications.

Forgotten marketing sites

A legacy microsite on a WordPress install the team forgot about.

Helpdesk / billing tools

Third-party support, chat or billing deployed as a self-hosted PHP app.

CI toolchains

A container used in CI, with PHP installed as part of a build step.

Vendor backends

A SaaS vendor's status page, docs portal or support widget on a PHP backend they manage.

THE "WE DON'T USE PHP" ANSWER

Most of the time, the person saying "we don't use PHP" is right about their own first-party code and wrong about the full inventory of PHP runtimes they're actually responsible for. Treat it as **an answer that requires evidence to accept, not one that closes the question.**

04

CHAPTER FOUR

Detection: what works, what doesn't

TL;DR

External scanning gives high-signal, high-coverage exposure reads — but can't mathematically guarantee safety for a runtime-level memory bug. The right posture pairs scanning with an infrastructure audit.

The honest starting position

"Black-box testing methods and even firewalls are not 100% accurate with this testing because it's still a very emerging vulnerability class right now."

— ANANDA KRISHNA, CTO, ASTRA SECURITY

That framing sets the right expectation for every scanner and WAF ruleset deployed against PHP2Shell, Astra's own included. External scanning gives a high-signal read on exposure. It cannot give a mathematical guarantee of safety — which is why the right posture pairs scanning with an infrastructure audit.

What external detection actually does

1

Version discovery

Fingerprint the PHP runtime behind the app and compare against the fixed list. The most reliable single signal — a patched runtime isn't vulnerable even if the app looks suspicious.

2

Targeted payload testing

Send crafted requests that probe whether the server-side configuration would accept the request shapes the exploit depends on — a reasonable proxy for exposure without attempting full exploitation.

3

Out-of-memory payload testing

Memory-pressure probes observe whether the target behaves consistently with a vulnerable runtime without crashing it — more sensitive than version discovery, catching obscured version strings and partial patches.

WHERE EXTERNAL DETECTION FALLS SHORT

A scanner can't see a runtime behind an API gateway that normalises unusual bodies, or one bundled in an internal-only container. It can't verify a patch actually took effect — a server may report a patched version string but still run the old binary because the processes **weren't restarted**. And it gives a point-in-time answer: clean on Tuesday can mean vulnerable on Thursday after a new deploy or restored backup.

What Astra's engines detect

Astra's DAST and Autonomous Pentest engines both detect PHP2Shell exposure using all three techniques above. Two scan types are purpose-built for this moment:

Emerging Threat Scan

Purpose-built for freshly-disclosed CVEs; spins up quickly against newly-affected version ranges.

Lightning Scan

A focused set of targeted test cases for high-severity issues, including PHP2Shell, as a fast first-line check.

Where WAFs fit

Cloudflare and others rolled out managed rules within hours. They protect against known request shapes, but the chunked-and-multipart primitives at the heart of PHP2Shell have many legitimate-looking variants. **Treat WAF rules as defence in depth and a bridge to patching — not a substitute for either.**

A CLASS SCANNERS CAN'T REACH

The same response calls covered a related Lite LLM issue affecting many apps using that library internally. "It is not always possible to detect it externally — so for that, audit your infrastructure to see where Lite LLM is being used." A reminder that infrastructure audit is the backstop for cases scanners simply cannot see.

05

CHAPTER FIVE

Exploitation in the wild

TL;DR

Attackers were probing before the advisory landed. POCs and scanners are now public. The question shifts from "will we be targeted?" to "were we touched during the pre-patch window?"

The pre-disclosure gap

The most unsettling data point about PHP2Shell is not the technical anatomy. It's Amazon's observation that exploitation attempts were visible in the MadPot sensor network **before** the public advisory. By the time CISA and NSA published on October 21, somebody had already been probing in the wild.

THE SHAPE OF THE QUESTION CHANGED

It stops being "will we be targeted?" and becomes "**were we touched during the window between the first attacker activity and our first patch?**" For most organisations that window is at least a week, and probably longer — the full timeline was known only to a small coordinated group during disclosure.

2004 - 2024

The long shadow. A 2004 finding, a 2020 bug report (#78876), and a 2024 OSTIF/Quarkslab audit that re-flagged it but didn't prove exploitation.

Aug 1, 2026

Discovery. Calif found the first exploitation path four days into OpenAI's Patch the Planet audit.

Aug 11 - 21

Coordinated disclosure. First bug reported to the PHP Foundation; POC confirmed Aug 15; second bug Aug 21.

Oct 21, 2026

Patches + press. Fixed versions shipped with CISA, NSA & WSJ/NYT coverage — and Amazon's pre-advisory finding.

The vulnerability class sat in the open for 22 years; attacker probing began inside the disclosure window.

Signals worth grepping for

Specific IPs and hashes should come from Amazon's MadPot advisory. Independent of those, these log signals are useful to look for:

- Multipart POST requests with unusually large, deeply-nested field names (patterns like `files[a][b][c][d]...`).
- Multipart requests with a declared `Content-Length` significantly smaller than the apparent decoded size.

- Chunked HTTP/1.1 or HTTP/2 lengthless requests to PHP endpoints.
- Repeated requests triggering PHP worker fatal errors — memory-limit or allocation failures clustering in time or by source IP.
- Session upload progress activity not matching any application feature.
- Outbound connections from PHP workers to unfamiliar destinations shortly after a matching inbound request.

None of these alone proves exploitation. Clusters of them, in the pre-patch window, warrant deeper investigation.

"WE HAVEN'T BEEN SCANNED YET" IS NOT REASSURANCE

Post-disclosure activity is broad, automated and opportunistic. A system that hasn't shown probing by Tuesday is almost certain to show it the following week as commodity tooling catches up to the public POCs. **Absence of evidence is not evidence of absence here.** The time-to-exploitation curve has compressed from weeks to hours.

06

CHAPTER SIX

Remediation done properly

TL;DR

Upgrade the runtime, not the application. Get on a fixed branch, handle EOL branches via backport or upgrade, apply mitigations only as a bridge, and verify the fix is actually running.

The five steps

1

Upgrade the runtime, not the application

The most common week-one mistake is patching WordPress/Drupal/dependencies and declaring done — this closes **zero** of the two CVEs. Update the PHP binary your apps execute against, and restart the processes using it.

2

Get on a fixed branch

Install **8.2.35** / **8.3.36** / **8.4.27** / **8.5.12** from your distribution's security channel where possible. From source, verify checksums against the PHP Foundation advisory and confirm the running binary matches the installed one after restart.

3

Handle unsupported branches

In order of preference: (a) distribution backport — trust the package manager's advisory, not `php -v`; (b) community-contributed patch, a last resort for regulated workloads; (c) accelerate the upgrade off the unsupported branch entirely — the right long-term answer.

4

Temporary mitigations (a bridge, not a fix)

These measures reduce specific attack paths, not the underlying vulnerabilities. Apply by CVE across relevant deployments.

CVE-2026-109619 · Stale global state

Set `session.upload_progress.enabled=0` in early-applied PHP configuration and restart all workers. On PHP 8.5, also set `fatal_error_backtraces=0`. Set `display_errors=0` in production as additional hardening. These measures do not cover every exploitation path.

CVE-2026-109618 · Out-of-memory corruption

Administratively enforce `memory_limit=-1` so applications cannot override it. This blocks a confirmed memory-limit failure path, not all allocation failures. Set `post_max_size=8M` without uploads or 32M with uploads, alongside appropriate request-size controls. Enforce container or cgroup limits, including `memory.max`, to manage memory-exhaustion risks. Avoid `RLIMIT_AS` and `RLIMIT_DATA`: they can trigger vulnerable allocation failures.

5

Verify the fix is actually running

Three checks in sequence — the step that catches real incidents.

STEP 5 · VERIFY IN SEQUENCE

1. Binary matches the fixed version. Run `php -v` ; cross-check distribution advisories for backports. **2. Processes use the new binary.** PHP-FPM pools and Apache workers don't auto-pick-up new binaries — confirm the process start time is after the package install time. **3. Behaviour matches the patched version.** Run a targeted CVE-specific scan. If any check fails, the system is still exposed, regardless of what the package manager says.

07

CHAPTER SEVEN

Post-exposure: assume you were a target

TL;DR

Patching does not rule out earlier compromise. Both CVEs can enable pre-auth RCE; CVE-2026-109619 can also expose memory. Investigate system access and rotate potentially exposed credentials in priority order.

Patching closes the door, not the question

A patched system is not a safe system. The patch stops future exploitation; it does not tell you whether exploitation already happened during the window between the first attacker activity and your patch. For most organisations, the honest working assumption is that **any PHP-dependent system reachable from the internet during Aug–Oct 2026 was a potential target.**

What PHP2Shell means for secrets and access

Exploitation can expose secrets or enable code execution with the PHP worker's privileges. Investigate credential access, modified files, unexpected processes and outbound connections. Rotate confirmed or suspected exposed credentials in the following priority order.

1

Payment processor keys

Stripe and equivalents — unauthorised use produces direct, immediate financial loss.

2

Cloud provider keys

AWS access keys, GCP service-account tokens, Azure principals.

3

LLM provider keys

Anthropic, OpenAI and similar — unauthorised use produces direct financial loss.

4

Database & cache credentials

Connection strings held in process memory for the duration of a request.

5

Session signing keys & HMAC secrets

More coordination needed — invalidating them logs out all active users.

If you find something

Host containment alone is insufficient. Isolate affected systems, preserve forensic evidence and rotate compromised credentials wherever valid. Investigate access gained with the PHP worker's privileges, including modified files, unexpected processes, persistence and lateral movement.

If you find nothing

A clean log review is good news, not proof of safety. The honest framing: the investigation reduced the probability of undetected exploitation to a level you're comfortable with — and sensitive credentials in play during the window still warrant rotation.

08

CHAPTER EIGHT

Compliance & regulatory obligations

TL;DR

PHP2Shell can involve code execution, data exposure or both. Assess notification obligations with counsel based on incident impact. Document what happened, when it became known and who was affected.

Assess compliance alongside the technical response

Investigate both system compromise and data exposure. Record affected systems, data, services and third parties. With counsel, determine applicable reporting obligations and when their notification clocks begin.

Frameworks that apply almost universally

Framework	Trigger & clock
GDPR / UK GDPR	72-hour notification to the supervisory authority after becoming aware of a personal-data breach likely to risk individuals' rights. "Becoming aware" is interpreted broadly.
NIS2 (EU)	Early warning within 24 hours , incident notification within 72, final report within one month. PHP2Shell meets the "significant incident" threshold.
DORA (EU financial)	ICT-incident reporting that overlaps and extends beyond NIS2; walk this path first for financial entities.
SEC Form 8-K (US public)	Item 1.05 — disclose a material incident within four business days of determining materiality. The clock starts on the materiality determination.
State laws (US)	All 50 states + DC; "without unreasonable delay" to specific day counts. CCPA/CPRA adds a private right of action.
HIPAA (US health)	Notify individuals within 60 days; HHS for breaches affecting 500+ individuals.
PCI-DSS	Notify acquirer and card brands on discovery of confirmed/suspected compromise — mandatory where card data was in PHP memory.

Asia-Pacific & beyond: DPDP (India), PDPA (Singapore, 72h), Privacy Act (Australia), PIPEDA (Canada), LGPD (Brazil, 72h), PIPL (China), plus PIPA (S. Korea), APPI (Japan), POPIA (S. Africa). The pattern is consistent even where timelines differ.

ATTESTATIONS & INSURANCE

SOC 2 / ISO 27001: undocumented response, missed in-scope systems, or failure to follow stated IR procedures becomes an audit finding. The remediation documentation created during your 72-hour response **is** the evidence auditors ask for. **Cyber insurance:** many policies require notifying the carrier within 24–72 hours; failure can void coverage. Put the carrier on the notification list alongside regulators.

WHEN THE CLOCK STARTS

Treat **October 21, 2026** as a **potential clock-start date** for every framework in scope, document your reasoning for the start date you choose, and get legal review before publishing — or declining to publish — any notification. Regulators accept defensible, documented judgement calls and reject undocumented ones even when the underlying decision was correct.

09

CHAPTER NINE

The supply-chain dimension

TL;DR

Auditing your own PHP does half the job. The other half lives in every third party that touches your data, credentials or infrastructure. Five questions structure the vendor conversation.

Why vendor risk is the real exposure

PHP2Shell is a supply-chain vulnerability in the fullest sense. The runtime hides inside SaaS products, hosting platforms, bundled appliances and legacy integrations. A vendor that processes your data, holds your keys, or connects to your production environment is a potential path to compromise — and the only way to understand that path is to ask.

The five questions below are structured in the order most useful for the first vendor conversation.

1

Do you use an affected PHP runtime?

Including bundled components and third-party-managed infrastructure. A thin "we don't use PHP" without evidence is a signal to escalate, not to accept.

2

Have you patched the actual PHP runtime?

Not the app or a plugin — the runtime underneath. Ask for the deployed version, evidence of the update, and confirmation that processes were restarted.

3

Could our data or credentials be exposed?

This requires a real investigation. A good answer is a written impact assessment separating confirmed from suspected from inconclusive.

4

What are you doing to detect activity?

Were logs reviewed for the pre-disclosure window signatures? Were they available for the period? What's the forward monitoring plan?

5

When can you consider the issue resolved?

Runtimes patched & verified, pre-patch exploitation investigated to an appropriate level, exposed credentials rotated and dependent services updated, residual risks documented, and a commitment to notify you if new findings change the assessment. If any are missing, the assessment stays open with a follow-up date.

WHAT GOOD VENDOR RESPONSES SHARE

They identify **specific** affected systems rather than making blanket statements; provide **evidence** of patching (versions, restart confirmation) rather than general assurances; and **acknowledge what they don't yet know**, especially about the pre-disclosure window, rather than claiming certainty they can't have. The worst responses show the opposite — and are worth escalating.

10

CHAPTER TEN

Response playbook: the first 72 hours

TL;DR

A time-boxed plan beats a general sense of urgency. Inventory, patch, validate, then rotate-communicate-document. A disciplined 72 hours produces a far better outcome than a frantic two weeks.

Four windows

Hour 0-4**Inventory everything running PHP**

Inventory PHP across production, staging, CI runners, legacy intranets, hosting panels, embedded consoles, container images and appliances. Record each runtime's version, deployment mode and exposure: internet-facing, internal or air-gapped. Include `mod_php`, PHP-FPM, LiteSpeed/LSAPI, `php-cgi`, FrankenPHP and CLI. Use this inventory to track the response.

Hour 4-24**Patch what you can, mitigate what you can't**

Patch supported branches first via distribution channels. Backport unsupported branches where available. Apply Chapter 6 mitigations where you can't patch in-window. Where even mitigation isn't possible on an exposed, sensitive system — take it offline per NSA guidance.

Hour 24-48**Validate and detect**

Three parallel tracks: **external validation** (scan patched systems to confirm they no longer respond to known shapes), **internal verification** (version + restart time against the deployment log), and **log review** from the Aug 1 pre-disclosure window onward.

Hour 48-72**Rotate, communicate, document**

Rotate high-priority credential classes by hour 72. Brief exec/legal/customer-facing teams and send the vendor questionnaire. Prepare compliance notifications in draft. Document what was done, found, and remains uncertain — able to answer an auditor six months out.

PAST HOUR 72

The work shifts to steady-state: vendor responses arrive over weeks, lower-priority rotations complete over the month, legacy systems get upgrade projects scoped. Keep the incident ticket open until each thread reaches a documented resolution. **A disciplined 72 hours produces a far better outcome than a frantic two weeks** — and the time pressure PHP2Shell creates is a good forcing function for the discipline.

11

CHAPTER ELEVEN

How Astra Security helps

TL;DR

Astra gives security teams a faster, more complete read on exposure than infrastructure audit alone — and validates that remediation actually took effect across the application surface.

Four capabilities, directly relevant

DAST & Autonomous Pentest

Both detect exposure using version discovery, targeted payload testing and OOM-based probing — covering surfaces easy to miss in an audit (marketing microsites, hosting panels, embedded admin consoles).

Emerging Threat & Lightning Scan

Built for exactly this pattern: a freshly-disclosed CVE, a short window, and a need for fast, high-confidence coverage without a full-weight pentest engagement.

Free PHP2Shell scanner

At php2shell.getastra.com — submit a URL, get a quick external check on whether the target responds to known PHP2Shell shapes. The intent is help, not pitch.

Autonomous Pentest for validation

Goes beyond detection to validate whether a weakness is actually exploitable in a specific app's context — closing the gap between "scanner says vulnerable" and "actually exposed".

THE HONEST FRAMING

Astra's engines are capable of detecting this vulnerability class. External scanning in general cannot guarantee 100% coverage for an emerging runtime-level bug — so the recommended posture is to **pair scanning with the infrastructure audit this ebook keeps returning to**. If this chapter is the only one you remember, the ebook has failed its job: the value lives in the eight chapters on the vulnerability and the response.

CHAPTER TWELVE

12 Checklist & board briefing reference

TL;DR

The one-page version: an audit checklist, a vendor questionnaire, the credential rotation order, and three board-ready paragraphs to lift straight into an update.

Infrastructure audit checklist

- ☐ Inventory all PHP hosts and deployments, including panels, intranets, embedded consoles, container images and web server integrations.
- ☐ Record PHP version, exposure and deployment mode: **mod_php**, PHP-FPM, LiteSpeed/LSAPI, **php-cgi**, FrankenPHP or CLI.
- ☐ Flag every host running an unfixed version for patching.
- ☐ Verify the installed PHP binary matches a fixed version (8.2.35 / 8.3.36 / 8.4.27 / 8.5.12).
- ☐ Verify all relevant PHP workers and long-running processes were restarted after the runtime update.
- ☐ Run an external scan (DAST, Autonomous Pentest, Emerging Threat, or the free scanner) against each patched system.
- ☐ For hosts you can't patch immediately, apply Chapter 6 mitigations or take offline per NSA guidance.
- ☐ Document every inventory entry with its final patched version and date of restart.

Credential rotation priority

1–3

Payment keys, then cloud keys (AWS/GCP/Azure), then LLM provider keys

4

Database & cache credentials

5

Session signing keys & HMAC secrets

Affected & fixed versions at a glance

8.5.x

8.5.12

8.4.x

8.4.27

8.3.x

8.3.36

8.2.x

8.2.35

Board briefing template

Three paragraphs the CISO can use as-is. Fill the organisation-specific numbers in square brackets.

01 — The incident

On October 21, 2026, the PHP Foundation disclosed two critical vulnerabilities (CVE-2026-109618 and CVE-2026-109619) in the PHP runtime, which powers approximately 70% of websites globally. CISA, NSA Cyber and major national CERTs issued advisories recommending immediate patching. Amazon confirmed exploitation attempts were observed in the wild **before** public disclosure. Our exposure spans [number] internal PHP installations and [number] vendor-managed systems identified as PHP-dependent.

02 — Our response

The security team completed an initial inventory within [X] hours and has patched [X]% of affected systems on supported branches. End-of-life systems that cannot be patched immediately have been isolated or are protected by WAF-level mitigations pending upgrade. Credential rotation is complete for high-priority categories and in progress for the remainder. A scoped vendor questionnaire has been sent to [number] third parties. Compliance notifications are being prepared under [GDPR / SEC 8-K / state breach notification / sector-specific regimes] with general counsel's involvement.

03 — Residual risk & ask

The pre-disclosure attacker activity observed by Amazon means our risk model assumes potential exposure during the window between first known probing and our patch deployment. Log review is in progress against published indicators of compromise. We will update the board at [date] with the results, final compliance posture, and vendor response status. The incremental investment required to complete the response is [budget line], and we are tracking overall business impact at [materiality assessment].

Twenty-two years is a long shadow.

A 2004 memory-safety bug that nobody published came back in 2026 as a pre-auth RCE reachable from a crafted HTTP request, usable against most of the applications that run large parts of the internet.

The response work is the same work defenders have always done: inventory, patch, verify, rotate credentials, review logs, interrogate vendors, document the gaps. The forcing function this time is a vulnerability that reaches so widely the honest answer to "am I affected?" is almost always **yes** — through at least one dependency, somewhere in the stack.

The teams that run this response carefully, close out the vendor conversations, and document the gaps will find that most of them are fine. The teams that patch WordPress, declare done, and move on will find out otherwise over the next few quarters.

Check your exposure in minutes

Submit a URL for a free external PHP2Shell check.

php2shell.getastra.com