

Exposed Websites Running Outdated and End of Life PHP Versions: Global Impact, Sector and Country Risk Analysis, Financial Losses, Pre Measures and the Complete Bithost Upgrade Blueprint

An in depth investigation into the October 2026 PHP2Shell vulnerabilities (CVE 2026 109618 and CVE 2026 109619), the December 2026 PHP 8.2 end of life cliff, why more than a third of PHP websites remain on unsupported branches, which industries and nations face the heaviest damage, and how organisations can scan with ZScanner, harden their servers and upgrade safely to PHP 8.4 and PHP 8.5.

Published by	Bithost Research and Security Engineering Team (bithost.in)
Publication date	October 2026 (Incorporating the 21 and 23 October 2026 PHP Security Advisories)
Featured security products	ZScanner (zscanner.bithost.in), CodeSecurity (bithost.in/codesecurity), EWSAF (ewsaf.bithost.in), Legac o Meter (legacometer.bithost.in), Bithost CERT (cert.bithost.in)
Primary audience	Chief Information Security Officers, CTOs, Engineering Heads, System Administrators, Compliance Officers and Business Leaders

Page format	A4 Standard (210 mm by 297 mm), Full Technical and Executive Edition
Classification	Public Distribution Actionable Security & Upgrade Guidance

Contents and Report Structure

Section / Chapter	Title
Opening Section	A Message from Bithost
Opening Section	About Bithost
Opening Section	How to Read This Report
Chapter 1	Executive Summary
Chapter 2	About This Report
Chapter 3	Why PHP Still Powers the Web
Chapter 4	The PHP Release Lifecycle and the End of Life Cliff
Chapter 5	How Websites Leak Their PHP Version to the World
Chapter 6	Anatomy of Recent PHP Vulnerabilities
Chapter 7	The CMS and Framework Supply Chain Problem
Chapter 8	Technical Effects of Running Exposed or Outdated PHP
Chapter 9	Business and Operational Effects on Organisations
Chapter 10	Sectors Most Affected by Exposed and Outdated PHP
Chapter 11	Country and Regional Exposure Analysis
Chapter 12	Deep Dive Sector Scenarios and Failure Patterns
Chapter 13	Detailed Country and Regional Risk Profiles
Chapter 14	Thirty Country PHP Exposure and Readiness Reference Directory
Chapter 15	What Organisations Stand to Lose: Financial and Strategic Impact
Chapter 16	Regulatory, Compliance and Cyber Insurance Exposure

Chapter 17	Regulatory and Security Standard Compliance Mapping
Chapter 18	Pre Measures and Immediate Emergency Playbook
Chapter 19	Comprehensive PHP Hardening Reference for Production
Chapter 20	Engineering Guide: How to Upgrade Legacy PHP Safely
Chapter 21	Complete php.ini Security Directive Reference
Chapter 22	Server Architecture Hardening Blueprints
Chapter 23	Web Application Firewall and Web Server Rulebook for PHP
Chapter 24	Framework by Framework Upgrade Playbooks
Chapter 25	Complete Code Migration Reference: PHP 5.6 and 7.x to PHP 8.4 and 8.5
Chapter 26	Automated CI/CD Security Gates for PHP Engineering Teams
Chapter 27	Post Compromise Forensics and Threat Hunting for PHP Servers
Chapter 28	Executive Briefing Pack: Questions Every Board Should Ask This Week
Chapter 29	How Bithost Helps Organisations Scan, Upgrade and Protect Their PHP Estate
Chapter 30	Operational Guide: Deploying the Bithost Security Suite Across Your PHP Estate
Appendix A	50 Point PHP Security and Upgrade Self Assessment Checklist
Appendix B	Glossary of Technical and Regulatory Terms
Appendix C	Primary Sources and References

A Message from Bithost

On 21 October 2026 the PHP project published a security advisory that most website owners will remember for a long time. Two flaws in the PHP engine itself, tracked as CVE 2026 109618 and CVE 2026 109619 and now known across the industry as PHP2Shell, allow an attacker with no password and no account to run code on a web server. In the most exposed setups the attack works before a single line of the website's own code runs. The PHP team asked every user to update immediately.

We wrote this report in the days that followed because the advisory exposed a problem that has been growing quietly for years. A large share of the world's websites still run PHP versions that no longer receive security fixes. When a flaw like PHP2Shell appears, the owners of those sites have no official patch to install. They are left with a choice between an emergency upgrade, a set of partial workarounds, or the hope that nobody notices them.

Hope is not a security control. Attackers do not need to find your organisation by name. They scan the whole internet, read the version banner your server sends back, and work through the list. Public scanning data collected after the disclosure showed that more than a million exposed PHP hosts sat on legacy or vulnerable configurations, and that almost none had been patched a full day after the fix shipped.

At Bithost we build security products and deliver security services for organisations of every size. Our vulnerability scanner ZScanner, our code analysis platform CodeSecurity, our Windows audit framework EWSAF, our legacy technology assessment Legac o Meter and the Bithost Certification Authority all exist for one reason. We want organisations to see their real risk clearly and to fix it before someone else uses it against them.

This report explains what outdated PHP means in plain language, who is exposed, which sectors and countries face the heaviest impact, what organisations stand to lose, and what they can do this week, this quarter and this year. The final chapter describes how Bithost can help you scan, upgrade and certify your PHP estate. Everything before that chapter is written to be useful whether or not you ever work with us.

Why this report matters now

PHP2Shell was disclosed on 21 October 2026. PHP 8.1 lost official support on 31 December 2025. PHP 8.2 will lose official support on 31 December 2026, which is about ten weeks from the date of this report. Any organisation running PHP 8.2 or older needs a plan in place before the end of the year.

Read it, share it with your technology and leadership teams, and use the checklists at the end. If you want a second pair of eyes on your estate, our team is ready to help.

The Bithost Research and Security Team

About Bithost

Bithost is an Indian technology and cybersecurity company. We design and build security software, run security assessments, and help organisations modernise the systems that carry their most important data. Our work sits where engineering meets security. We believe a secure system has to be a maintainable system, and that most security failures begin as maintenance failures.

What we build

Product	What it does	Where to find it
ZScanner	Desktop vulnerability scanner with 19 scan modules covering more than 40 vulnerability classes. Runs fully offline and maps findings to OWASP Top 10 2021, OWASP API Security Top 10 2023 and PCI DSS 4.0.	zscanner.bithost.in
CodeSecurity	Self hosted static analysis, software composition analysis, secret scanning and infrastructure as code checks in one lightweight application. Generates CycloneDX software bills of materials.	bithost.in/codesecurity
EWSAF	Enterprise Windows Security Audit Framework. Read only and agentless, with 60 audit modules mapped to CIS, NIST, ISO 27001, PCI DSS, HIPAA and SOC 2.	ewsaf.bithost.in
Legacy Meter	Free legacy technology risk assessment for technology leaders. A 25 question diagnostic produces a weighted score, a letter grade and a board ready PDF.	legacometer.bithost.in
Bithost Certification Authority	Certification for DPDPA 2023, GDPR, ISO 27001 readiness and Cyber Trust, with verifiable certificates and a public trust registry.	cert.bithost.in

Table 0.1: The Bithost product family

What we do

- Vulnerability assessment and penetration testing for web applications, APIs and infrastructure.
- PHP version upgrades and application modernisation, from legacy PHP 5 and PHP 7 code to PHP 8.4 and PHP 8.5.
- Secure code review and software supply chain assessment.
- Compliance readiness for India's Digital Personal Data Protection Act 2023, CERT In directions, PCI DSS and ISO 27001.
- Incident response support and post incident hardening.

You can reach us through bithost.in. Every product listed above offers either a free tier, a free assessment or a trial, so you can start without a purchase decision.

How to Read This Report

This report runs to more than seventy pages because the subject touches technology, business, law and geography at once. Not every reader needs every chapter. The guide below points each reader to the parts that matter most to them.

If you are	Start with	Then read
A board member or chief executive	Executive Summary	Chapters 9, 12 and 13 on effects, losses and regulation
A chief information security officer	Executive Summary and Chapter 6 on PHP2Shell	Chapters 8, 14, 15 and 17
An IT or engineering manager	Chapter 4 on the release lifecycle	Chapters 14, 15 and 16 on prevention and upgrades
A developer or system administrator	Chapter 6 and Chapter 14	Chapter 15 hardening and the appendices
A compliance or legal officer	Chapter 13 on regulation	Chapters 9 and 12
A small business owner	Executive Summary	Chapter 14 and the self assessment checklist in Appendix A

Table 0.2: Reading guide by role

Conventions used

- End of life, written EOL in tables, means the PHP project no longer publishes fixes of any kind for that branch.
- Supported means the branch still receives at least security fixes from the PHP project.
- Figures drawn from third party research are attributed to their source in the text and listed in the references.

- Figures marked as a Bithost assessment are our own analysis. We explain the method behind each one.
- Configuration examples are shown in a fixed width font and can be copied into the relevant file.
- All dates use the day month year format. The research cut off date for this edition is 24 October 2026.

Chapter 1. Executive Summary

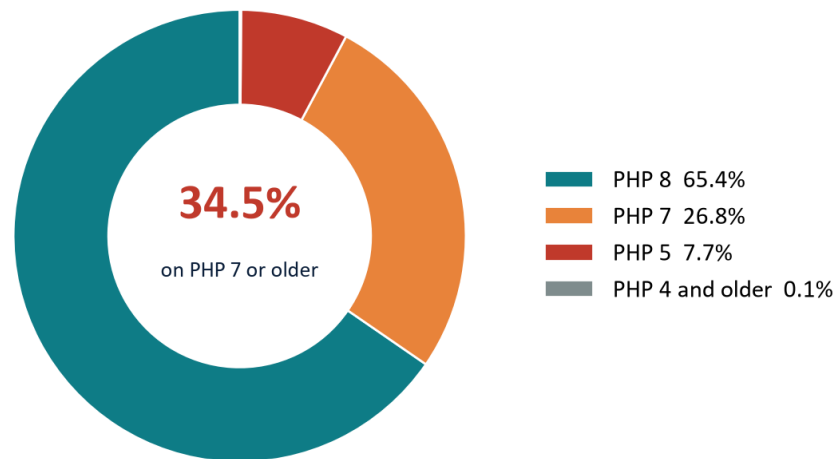
PHP is the language behind most of the websites on the internet. It runs WordPress, Drupal, Joomla, Magento, Moodle, countless government portals and an enormous number of custom business applications. It is mature, fast in its modern versions and free. Its success is also the source of the risk described in this report. Because PHP is everywhere, old copies of PHP are everywhere too.

1.1 The situation in October 2026

The PHP project supports each release branch for four years. Two years bring bug fixes and security fixes. The final two years bring security fixes only. After that the branch reaches end of life and receives nothing. As of 24 October 2026 the only supported branches are PHP 8.2, 8.3, 8.4 and 8.5. PHP 8.2 drops out of support on 31 December 2026.

Public survey data from W3Techs reported in October 2026 shows PHP 8 running on 65.4 percent of PHP websites, PHP 7 on 26.8 percent and PHP 5 on 7.7 percent. Every copy of PHP 7 and PHP 5 is end of life. Many copies of PHP 8.0 and PHP 8.1 are end of life as well. A third of PHP websites, and therefore a very large slice of the whole web, run code that the people who wrote it have stopped fixing.

Major PHP Versions Among PHP Websites



Source: W3Techs web technology survey, October 2026 (as publicly reported)

Figure 1.1: Major PHP versions among PHP websites, October 2026

1.2 PHP2Shell changed the urgency

On 21 October 2026 the PHP project released PHP 8.5.12, 8.4.27, 8.3.36 and 8.2.35 to fix two critical flaws that allow unauthenticated remote code execution. The advisory states that every PHP installation serving web requests is at risk, including PHP FPM, Apache with mod_php, PHP CGI, FrankenPHP and LiteSpeed. The PHP team named mod_php as the highest risk because a demonstrated exploit triggers before any application code runs. Amazon Threat Intelligence and other monitoring teams reported probing activity before the public disclosure.

Censys reported 36.7 million internet facing service records with potentially affected PHP footprints. Shodan based analyses published by OX Security and Black Kite counted between 1.9 and 2.2 million measurable PHP hosts, with 1.37 million on legacy or vulnerable configurations. OX Security found that 99.3 percent of measurable instances were still unpatched 24 hours after the fix was released.

Because of the severity, the PHP security team also published minimal source backports for PHP 7.4, 8.0 and 8.1. Those backports help organisations that build PHP from source. They do not help the large population of servers that run PHP 7.3, 7.2, 7.1, 7.0 or PHP 5, and they are not a substitute for running a supported branch.

1.3 Who is most exposed

Our sector analysis places government and public services, education, healthcare, retail and ecommerce, and small and medium businesses at the top of the risk table. These sectors combine a heavy legacy PHP footprint, sensitive data and slow upgrade cycles. Banking and fintech holds sensitive data but upgrades faster and operates under stricter supervision, which lowers its overall position.

By volume of exposed infrastructure, the United States, China and Germany lead the world because they host the largest share of servers. By combined risk, which weighs exposure against the ability to patch and upgrade quickly, India, Brazil, Indonesia and Vietnam stand out. These countries have large and fast growing PHP estates, a high proportion of small hosting customers on shared servers, and a long tail of legacy government and education portals.

1.4 What is at stake

IBM's Cost of a Data Breach Report 2025 puts the global average cost of a breach at USD 4.44 million. In the United States the average reached USD 10.22 million. In India it reached an all time high of INR 220 million. Healthcare remained the most expensive sector for the twelfth year running at USD 7.42 million per breach. Sophos found that exploited vulnerabilities were the leading technical root cause of ransomware attacks for the third year in a row, accounting for 32 percent of cases.

The losses go beyond the incident itself. Organisations face regulatory penalties, which under India's Digital Personal Data Protection Act 2023 can reach INR 250 crore for a failure to take reasonable security safeguards. They face customer churn, search engine blacklisting, payment card compliance failures, higher insurance premiums and weeks of disrupted operations.

1.5 What organisations should do

1. Within 24 hours, identify every server and container running PHP, including third party appliances and hosted platforms.
2. Within 72 hours, update every supported branch to PHP 8.5.12, 8.4.27, 8.3.36 or 8.2.35 or later.
3. On any host that cannot be updated at once, apply the interim mitigations published by the PHP project and place the host behind a web application firewall.
4. Within 30 days, scan every PHP application for vulnerabilities and build a ranked upgrade plan.
5. Within 90 days, move every application to PHP 8.4 or PHP 8.5, and retire or isolate anything that cannot move.
6. Put a standing process in place so that no PHP branch ever reaches end of life inside your estate again.

1.6 Key figures at a glance

Measure	Figure	Source
Share of websites with a known server side language that use PHP	69.3 percent	W3Techs, October 2026
PHP websites on PHP 7 or older	34.5 percent	W3Techs, October 2026
Internet facing services with potentially affected PHP	36.7 million	Censys, October 2026
Measurable PHP hosts in Shodan analyses	1.9 to 2.2 million	OX Security and Black Kite
Hosts on legacy or vulnerable configurations	1.37 million	Shodan based analysis
Instances unpatched 24 hours after the fix	99.3 percent	OX Security
Global average cost of a data breach	USD 4.44 million	IBM, 2025
Average cost of a data breach in India	INR 220 million	IBM, 2025
Ransomware attacks rooted in exploited vulnerabilities	32 percent	Sophos, 2025

Measure	Figure	Source
Breaches starting with vulnerability exploitation	20 percent	Verizon DBIR 2025
Date PHP 8.2 reaches end of life	31 December 2026	php.net

Table 1.1: Key figures

Chapter 2. About This Report

2.1 Purpose

This report has three goals. The first is to give an accurate picture of how many websites run outdated PHP and why that matters after PHP2Shell. The second is to explain the effects in terms that a business leader can act on, including which sectors and countries carry the most risk and what the financial exposure looks like. The third is to give practical, tested steps that any organisation can follow to reduce that risk, starting the same day.

2.2 Scope

The report covers PHP as a server side runtime used to serve web requests. It includes PHP running under Apache with mod_php, PHP FPM behind Nginx or Apache, PHP CGI, LiteSpeed, FrankenPHP and PHP inside containers and managed hosting platforms. It covers applications built on PHP frameworks and content management systems as far as their exposure depends on the underlying PHP version.

The report does not attempt a complete review of every plugin or theme vulnerability in WordPress or other platforms. Those are important, and Chapter 7 touches on them, but the core subject is the PHP runtime and the risk of running a version that no longer receives fixes.

2.3 Method

We combined four kinds of evidence. Official sources from the PHP project gave us release dates, support windows and the full text of the October 2026 advisory. Internet measurement data from W3Techs, Censys and Shodan based analyses gave us the size and shape of the exposed population. Industry research from IBM, Verizon and Sophos gave us the cost and frequency of breaches. Our own assessment, built from those sources and from Bithost field experience, gave us the sector and country risk rankings.

Evidence type	Sources used	How we used it
Official project data	php.net supported versions page, end of life table and security advisories of 21 and 23 October 2026	Release lifecycle, affected versions, fixed versions and official mitigations
Web technology surveys	W3Techs survey of the top ten million websites	Share of PHP and share of each major PHP version
Internet scanning	Censys, Shodan based analyses by OX Security and Black Kite	Count of exposed services and hosts and patch rates after disclosure

Evidence type	Sources used	How we used it
Threat intelligence	Amazon Threat Intelligence, GreyNoise, Akamai, Bitdefender reporting	Exploitation timing, attacker behaviour and target geography
Breach economics	IBM Cost of a Data Breach 2025, Sophos State of Ransomware 2025, Verizon DBIR 2025	Average losses, root causes and attack vectors
Regulation	Primary texts of DPDPA 2023, CERT In directions, GDPR, NIS2, PCI DSS 4.0	Penalties, reporting duties and patch requirements
Bithost assessment	Analysis of the above plus Bithost engagement experience	Sector heatmap, country matrix and loss models

Table 2.1: Evidence used in this report

2.4 Limits of the data

Every internet measurement has blind spots, and readers should keep three in mind. First, scanners can only count servers that reveal their PHP version. Many servers hide the version header, so the real number of outdated installations is higher than any scan shows. Second, a version banner does not always tell the full story. Some Linux distributions backport security fixes into older version numbers, so a server reporting PHP 7.4 may carry some fixes. Third, survey figures change daily. The numbers in this report reflect the position in October 2026 and readers should check the live sources for current values.

W3Techs restricts automated access to its full pages, so the version shares in this report are taken from its October 2026 figures as publicly reported. Third party exposure counts are presented as each organisation published them. Where two sources give different figures, we show the range.

2.5 A note on language

We have written this report for people who run organisations as well as people who run servers. Where a technical term is unavoidable we explain it the first time it appears, and Appendix C contains a glossary. We have avoided vague wording. When we state a figure we name its source. When we give an opinion we say that it is ours.

Chapter 3. Why PHP Still Powers the Web

Every few years a commentator predicts the death of PHP. Every few years the public web ignores that prediction. According to W3Techs survey data in October 2026, PHP runs on 69.3 percent of all websites whose server side programming language is known. No other server side runtime comes close in deployed site count. Understanding why outdated PHP is such a widespread problem begins with understanding how PHP became the default foundation of the internet.

3.1 The economics of shared and managed hosting

For more than twenty five years, PHP has been the only dynamic language installed by default on almost every inexpensive web hosting account in the world. A small business in Pune, a municipal office in Sao Paulo, a clinic in Jakarta or a school in Warsaw could buy a hosting plan for a few dollars a month, upload a folder of files over FTP or through a control panel such as cPanel or Plesk, and have a working website the same afternoon. Node, Python, Go and Java typically required virtual private servers, process managers or container pipelines that cost more money and demanded specialist staff.

That economic advantage created a huge installed base of websites owned by organisations that do not have a full time system administrator. A local agency or freelance developer built the site, handed over the login credentials and moved on to the next client. Five or eight years later the business still relies on the website for orders, bookings or citizen enquiries, while the underlying PHP runtime remains frozen on whatever version the hosting account defaulted to when the contract started.

3.2 Content management systems and ecommerce platforms

PHP's footprint is inseparable from the platforms written in it. WordPress alone powers more than forty percent of all websites on the internet. Drupal runs national government portals, university networks and large publishing groups. Joomla continues to serve tens of thousands of public sector and community portals. Magento and its community fork OpenMage, along with WooCommerce, PrestaShop and OpenCart, process billions of dollars in retail transactions every year. Moodle sits at the centre of online learning for universities, schools and corporate training departments worldwide.

Platform	Primary use case	Typical PHP risk pattern
WordPress and WooCommerce	Corporate sites, media publishers, small and midsize ecommerce	Core supports modern PHP, but abandoned third party plugins break when administrators switch from PHP 7.4 to PHP 8.x
Drupal	Government portals, universities, large enterprises	Sites left on Drupal 7 or early Drupal 9 remain pinned to older PHP

Platform	Primary use case	Typical PHP risk pattern
		branches until a funded migration project takes place
Joomla	Public sector sites, associations, regional businesses	Long lived templates and custom extensions often call functions removed in PHP 8.0
Magento and Adobe Commerce	Midsize and enterprise retail stores	Complex checkout customisations and payment modules require careful regression testing before any PHP minor upgrade
Moodle	Schools, universities, vocational training academies	Academic calendars allow upgrade windows only twice a year, and custom grading plugins delay runtime updates
Laravel and Symfony applications	Custom business portals, SaaS backends, internal tools, APIs	Framework upgrades are tied to Composer dependency trees that lock specific PHP version constraints
CodeIgniter, CakePHP and custom PHP	Line of business apps, billing systems, regional portals	Many applications were written between 2010 and 2018 with direct database calls and no automated test suite

Table 3.1: Major PHP platforms and why they stay on older runtimes

3.3 The plugin and dependency trap

When a site owner clicks the PHP version selector in a hosting panel and changes PHP 7.4 to PHP 8.4, the core content management system usually works without trouble. What fails is a single commercial plugin bought six years ago from a developer who is no longer in business, or a custom theme built by an agency that never handed over documentation. PHP 8.0 made type handling much stricter and turned dozens of quiet warnings from PHP 7.4 into fatal errors that stop page rendering completely.

Faced with a blank white screen on their live checkout page, most site owners do the obvious thing. They switch the hosting selector back to PHP 7.4, confirm that the website loads again, and promise themselves they will sort out the broken plugin next month. Next month turns into next year, and a temporary rollback becomes a permanent exposure.

3.4 How PHP runs in production

PHP is not a single server program. It connects to web servers through a Server Application Programming Interface, known as a SAPI. The choice of SAPI affects both performance and how a vulnerability in the PHP engine behaves when an attacker sends a malicious request.

SAPI deployment mode	How it works	Security and exposure profile
Apache with mod_php	The PHP interpreter is loaded directly inside each Apache worker process	Highest risk in CVE 2026 109618 and CVE 2026 109619 because engine parsing happens inside the web server process before application code runs
PHP FPM behind Nginx or Apache	Nginx or Apache handles HTTP traffic and passes PHP requests over FastCGI to a pool of worker processes	Most common modern setup. Worker processes handle many requests one after another, so stale state in one worker can leak into the next request
PHP CGI on Windows or Unix	The web server launches a PHP CGI process for incoming requests using command line arguments and environment variables	Historically prone to argument injection flaws such as CVE 2024 4577 on Windows locales, especially inside XAMPP and WAMP stacks
LiteSpeed LSAPI	High speed integration used by LiteSpeed Web Server and OpenLiteSpeed on shared hosting platforms	Widely deployed across mass market hosting providers. Affected by core engine flaws alongside PHP FPM
FrankenPHP and Caddy	Modern Go based application server embedding the PHP interpreter for worker and standard modes	Growing adoption in containerised deployments. Requires rebuilding container images whenever a PHP security release ships

Table 3.2: PHP server architectures and their risk profiles

This diversity matters during an emergency. Updating PHP on a single virtual machine running Ubuntu and PHP FPM takes a few minutes if packages are current. Updating PHP across four hundred Docker images, thirty cPanel servers with MultiPHP, twelve Windows servers running XAMPP in branch offices and five legacy appliances where PHP is compiled into vendor firmware is an engineering project.

Chapter 4. The PHP Release Lifecycle and the End of Life Cliff

Every branch of PHP follows a predictable four year schedule set by the PHP Group. For the first two years after its initial stable release, a branch receives active support. Bugs and security flaws reported by the community are fixed and published in regular point releases. For the next two years, the branch enters its security fixes only phase. During those two years the PHP team publishes updates only when critical security issues are found. Once the four years end, the branch reaches end of life.

End of life is a hard boundary. The upstream PHP project stops releasing source packages, Windows binaries and security patches for that branch. When a new vulnerability is discovered in the PHP engine, researchers test it against supported branches, the PHP team patches the supported branches, and anyone still running an older branch is left exposed to a publicly documented flaw.

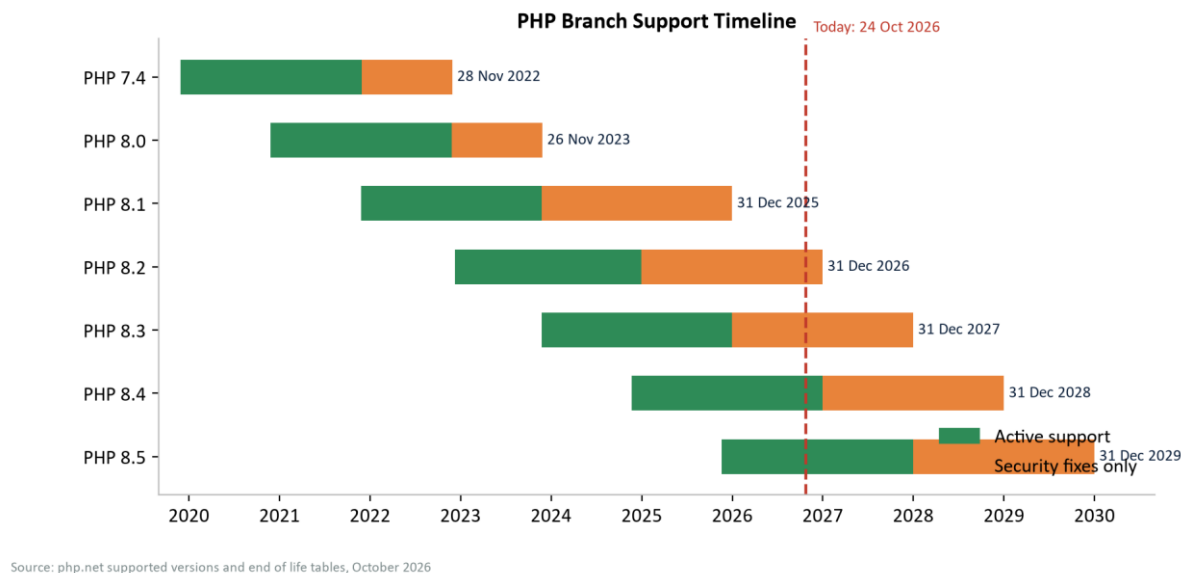


Figure 4.1: PHP branch support timeline as of 24 October 2026

4.1 Official branch status in October 2026

The table below records the official support schedule from php.net on 24 October 2026 together with the historical end of life dates for earlier branches that still appear in internet scans.

Branch	Initial release	Active support ended	Security support ends	Status on 24 Oct 2026	Patched release for PHP2Shell
PHP 5.6	28 Aug 2014	31 Dec 2016	31 Dec 2018	End of life for nearly 8 years	None from upstream
PHP 7.0 to 7.3	Dec 2015 to Dec 2018	Dec 2017 to Dec 2020	Dec 2018 to Dec 2021	End of life	None from upstream

Branch	Initial release	Active support ended	Security support ends	Status on 24 Oct 2026	Patched release for PHP2Shell
PHP 7.4	28 Nov 2019	28 Nov 2021	28 Nov 2022	End of life for nearly 4 years	Minimal source backport only
PHP 8.0	26 Nov 2020	26 Nov 2022	26 Nov 2023	End of life for nearly 3 years	Minimal source backport only
PHP 8.1	25 Nov 2021	25 Nov 2023	31 Dec 2025	End of life since Dec 2025	Minimal source backport only
PHP 8.2	8 Dec 2022	31 Dec 2024	31 Dec 2026	Security fixes only, 2 months left	PHP 8.2.35
PHP 8.3	23 Nov 2023	31 Dec 2025	31 Dec 2027	Security fixes only	PHP 8.3.36
PHP 8.4	21 Nov 2024	31 Dec 2026	31 Dec 2028	Active support	PHP 8.4.27
PHP 8.5	20 Nov 2025	31 Dec 2027	31 Dec 2029	Active support	PHP 8.5.12

Table 4.1: PHP version support status and October 2026 emergency patch versions

The December 2026 cliff for PHP 8.2

Many organisations upgraded from PHP 7.4 to PHP 8.1 or PHP 8.2 during 2023 and treated the job as finished. PHP 8.1 reached end of life on 31 December 2025. PHP 8.2 reaches end of life on 31 December 2026. Any team starting an upgrade project today should target PHP 8.4 or PHP 8.5 directly so they do not have to repeat the migration in a few months.

4.2 The myth of operating system backports

System administrators often point out that enterprise Linux distributions such as Red Hat Enterprise Linux, Ubuntu LTS and Debian backport security fixes into the older PHP package versions shipped with that operating system release. When that mechanism works, a server whose banner says PHP 7.4 or PHP 8.1 on a supported Ubuntu LTS release may in fact contain patches for newer CVEs.

In practice, our field audits uncover four recurring problems with relying on distribution backports alone.

1. Many servers do not get their PHP packages from the official distribution repository at all. They use third party repositories, custom source builds, XAMPP installers, old Docker Hub images or control panel bundles whose maintainers stopped updating that branch when upstream support ended.
2. Automatic unattended upgrades are frequently turned off on production web servers because administrators fear an unexpected service restart during business hours.

3. PHP FPM or Apache worker processes must be restarted after a library or binary update. We regularly inspect servers where the patched package was unpacked on disk weeks earlier, while the running worker processes in memory are still executing the old vulnerable binary.
4. When the operating system release itself reaches end of regular support, such as Ubuntu 18.04 or CentOS 7, the distribution backports stop too unless the organisation pays for an extended lifecycle subscription and configures the private repository on every host.

Chapter 5. How Websites Leak Their PHP Version to the World

An attacker does not need to guess whether a web server runs PHP 5.6, PHP 7.4 or PHP 8.5. By default, PHP announces its exact version number on every single HTTP response. Internet scanning platforms such as Shodan and Censys visit every reachable IPv4 address and millions of domain names, record those response headers and index them into searchable databases.

5.1 The X Powered By response header

Out of the box, PHP sets the configuration directive `expose_php` to `On` in `php.ini`. Whenever PHP generates a page, it appends a header such as `X Powered By: PHP/7.4.33` to the HTTP response. Below is a typical response captured from an unhardened server.

```
HTTP/1.1 200 OK
Date: Sat, 24 Oct 2026 09:15:02 GMT
Server: Apache/2.4.41 (Ubuntu) PHP/7.4.3
X-Powered-By: PHP/7.4.3
Set-Cookie: PHPSESSID=8f3a9d1c7b2e4f60a1c3e5d7f9b1a3c5; path=/
Content-Type: text/html; charset=UTF-8
```

In six lines of plain text, that server has told every scanner on earth its operating system family, its Apache version, its exact PHP version down to the patch level, and the fact that it uses standard PHP session cookies.

5.2 Ten ways an application reveals its PHP stack

Exposure channel	What an external observer sees	Why it happens
X Powered By HTTP header	Exact string such as <code>PHP/7.4.33</code> or <code>PHP/8.1.12</code>	The <code>expose_php</code> directive defaults to <code>On</code> in standard <code>php.ini</code> files
Server HTTP header	<code>Apache/2.4.x (Unix) PHP/5.6.40</code>	Apache <code>mod_php</code> with <code>ServerTokens</code> set to <code>Full</code> or OS appends module versions
Forgotten phpinfo pages	Complete dump of PHP build date, loaded extensions, environment variables and server paths	Developers create <code>info.php</code> , <code>test.php</code> or <code>phpinfo.php</code> during setup and forget to delete it
Session cookie name	Set Cookie header containing <code>PHPSESSID</code>	Default <code>session.name</code> in <code>php.ini</code> is never changed
Unhandled error traces	Fatal error: Uncaught TypeError in <code>/var/www/html/app/Controller.php</code> on line 42	<code>display_errors</code> is left <code>On</code> in production instead of logging errors privately

Exposure channel	What an external observer sees	Why it happens
Composer files in web root	Publicly downloadable /composer.json and /composer.lock	Application root rather than the /public subfolder is configured as the web server document root
Exposed .env and config backups	Readable /.env, /wp config.php.bak or /config.php.swp	Editors leave backup files that do not end in .php, so the web server sends them as plain text
URL extensions and query patterns	Links ending in index.php, login.php or admin.php	Rewrite rules are not configured to hide file extensions
Docker and container registries	Public image manifests showing FROM php:7.4 apache	Organisations push build metadata or public images without stripping layer history
CMS generator tags and REST APIs	WordPress or Drupal version headers that imply minimum and maximum compatible PHP versions	Default CMS metadata headers are left enabled

Table 5.1: Common ways websites expose their PHP version and configuration

5.3 Why hiding the banner is helpful but never enough

Turning `expose_php` off stops casual search queries on Shodan from listing your server under a specific version number. Every server administrator should do it, and Chapter 15 shows the exact configuration line. However, hiding the header does not fix the underlying vulnerability. During mass exploitation waves, automated bots do not bother checking the version banner first. They fire the exploit payload at every web server on port 80 and port 443 and let the vulnerable ones execute it.

Chapter 6. Anatomy of Recent PHP Vulnerabilities

Nothing illustrates the danger of running unpatched or end of life PHP better than the vulnerabilities disclosed between 2024 and October 2026. These are not bugs in a poorly written third party plugin. They are flaws in the core PHP runtime itself, meaning that even a cleanly written application with parameterised database queries and strict input validation can be compromised when the engine beneath it is vulnerable.

6.1 PHP2Shell: CVE 2026 109618 and CVE 2026 109619

On 21 October 2026 the PHP project released an urgent security advisory along with patched versions PHP 8.5.12, 8.4.27, 8.3.36 and 8.2.35. Two days later, on 23 October 2026, the project published a follow up informational advisory regarding session upload progress and a related vector tracked for the regular cycle as CVE 2026 115560. Security researchers and the press quickly referred to the October 21 pair as PHP2Shell because a crafted HTTP request can take an attacker straight from the public internet to code execution on the server.

Identifier	Vulnerability class	CVSS score	Core technical mechanism	Fixed upstream versions
CVE 2026 109618 (GHSA w8hw w9fx j79w)	Remote code execution via out of memory corruption	9.2 Critical	Out of memory handling can interrupt PHP while internal request data is only partially initialized. During request cleanup, PHP may treat that invalid uninitialized data as valid memory structures.	8.5.12, 8.4.27, 8.3.36, 8.2.35 (plus minimal source backports for 7.4, 8.0, 8.1)
CVE 2026 109619 (GHSA 6wmf 4rv5 p3rp)	Remote code execution via stale global state	9.2 Critical	Request owned internal state can survive after its backing memory has been freed. A subsequent request handled by the same worker process may use that stale state before normal request initialization resets it.	8.5.12, 8.4.27, 8.3.36, 8.2.35 (plus minimal source backports for 7.4, 8.0, 8.1)
CVE 2026 115560 (advisory note 23 Oct 2026)	Session upload progress serialize handler vector	Pending regular release	Occurs when session.serialize_handler is set to php_binary or php_serialize in config and switched to php at runtime via ini_set() before session_start() while session.upload_progress.enabled is active.	Mitigated by session.upload_progress.enabled=0

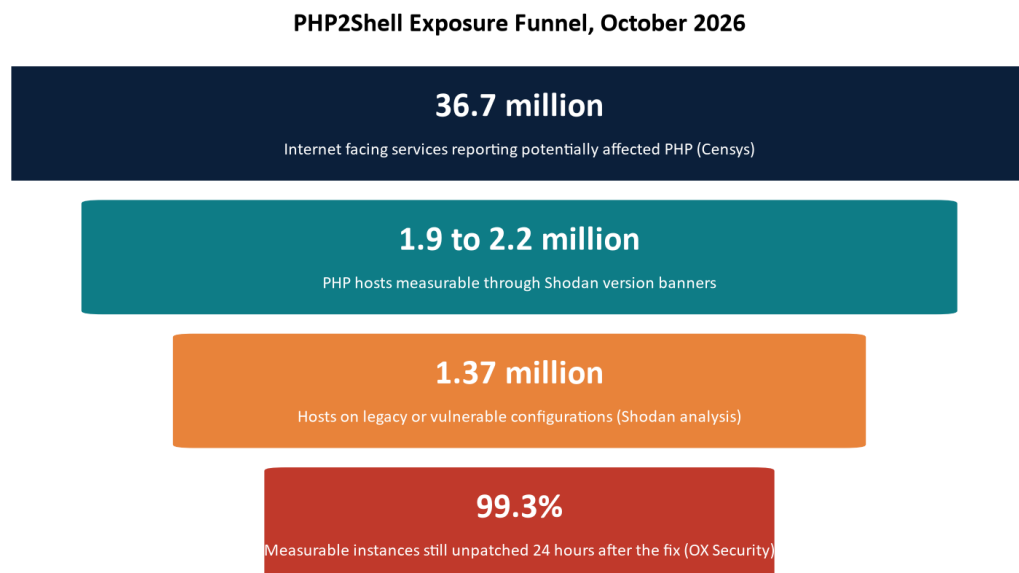
Table 6.1: Summary of the October 2026 PHP security advisories

Why PHP2Shell is so dangerous

Most web application vulnerabilities require the attacker to reach a specific piece of application code, such as a login form, a file upload controller or a search query. According to the official PHP advisory of 21 October 2026, all PHP installations serving web requests are at risk, including PHP FPM, Apache with mod_php, php cgi, FrankenPHP and LiteSpeed. For Apache with mod_php, the PHP project noted a demonstrated exploit where an unauthenticated request triggers the attack before any application code runs at all.

Let us unpack what happens in both flaws in plain terms.

- In CVE 2026 109618, PHP starts setting up internal data structures to handle an incoming HTTP request. Before setup finishes, a carefully sized payload forces PHP to hit its memory limit and jump into its out of memory bailout routine. The cleanup routine assumes the request structures were fully set up, reads whatever bytes were sitting in memory, and acts on pointers that the attacker has shaped.
- In CVE 2026 109619, a worker process handles Request A, which triggers an error or upload progress state that frees a block of memory without clearing the global pointer that refers to it. When Request B arrives at the very same worker process a millisecond later, PHP touches that leftover pointer before resetting the worker's state for the new request. That allows the attacker to hijack execution or read sensitive memory left behind by other users.



Sources: Censys, Shodan based analyses by OX Security and Black Kite, October 2026

Figure 6.1: Global exposure and first day unpatched rate for PHP2Shell, October 2026

The exposure numbers behind PHP2Shell

Within hours of the advisory, internet intelligence firms measured how many servers on the public internet were exposed. Censys identified approximately 36.7 million internet facing service records reporting potentially affected PHP footprints. Shodan analyses conducted by OX Security and Black Kite tracked between 1.9 and 2.2 million measurable PHP hosts exposing version details, with 1.37 million sitting on legacy or vulnerable configurations. Around 69 to 73 percent of version revealing hosts were running end of life branches that do not receive normal automated updates. Most striking of all, OX Security's twenty four hour post disclosure scan found that 99.3 percent of measurable instances remained unpatched after the first full day.

6.2 CVE 2024 4577: The Windows PHP CGI argument injection flaw

Two years before PHP2Shell, in June 2024, researchers disclosed CVE 2024 4577, rated CVSS 9.8 Critical. The flaw affected PHP running in CGI mode on Windows, particularly when Windows was configured for Traditional Chinese, Simplified Chinese or Japanese locales, and affected XAMPP installations out of the box. Because of Windows Best Fit character conversion, an attacker could send a soft hyphen byte, hexadecimal 0xAD, in the HTTP query string. Windows converted that byte into a standard ASCII hyphen, 0x2D, before passing it to php cgi.exe. The PHP binary interpreted the converted character as a command line switch such as `allow_url_include=1` and `auto_prepend_file=php://input`, running whatever PHP code the attacker placed in the request body.

Threat actors began scanning for CVE 2024 4577 within twenty four hours of disclosure. GreyNoise recorded more than a thousand unique malicious IP addresses still actively probing for the bug in January 2025 alone. Attackers used it to deploy TellYourPass ransomware, Gh0st RAT, Cobalt Strike beacons and RedTail cryptominers across Taiwan, Hong Kong, Japan, the United States, Brazil and India. The lesson was clear. Even an old deployment mode like CGI remains widespread in real enterprise networks, and attackers keep exploiting a high severity PHP flaw for years after the patch is released.

6.3 Other major PHP runtime vulnerabilities

CVE ID	Year	Affected component	What went wrong
CVE 2026 109618	2026	Core engine OOM handling	Out of memory interruption during partial request initialization leads to unauthenticated RCE
CVE 2026 109619	2026	Core engine request state	Stale global state survives after backing memory is

CVE ID	Year	Affected component	What went wrong
			freed, enabling RCE across worker requests
CVE 2024 4577	2024	PHP CGI on Windows	Windows Best Fit character mapping of soft hyphen 0xAD allows command line argument injection and RCE
CVE 2024 8926 and 8927	2024	PHP CGI and FastCGI	Bypass and configuration parsing flaws following the initial CGI hardening
CVE 2024 2756	2024	Cookie handling (__Host and __Secure)	Improper mangling of cookie names allowed attackers to forge security prefixed cookies
CVE 2023 3824	2023	Phar archive reading	Buffer mismanagement when reading phar directory entries could lead to remote code execution
CVE 2022 31629	2022	HTTP cookie parsing	Network attacker on a shared domain could inject standard cookies into secure ones
CVE 2019 11043	2019	PHP FPM with Nginx	Buffer underflow in env_path_info when Nginx fastcgi_split_path_info used a newline regex, widely exploited against PHP 7.x

Table 6.2: Selected high impact PHP runtime vulnerabilities, 2019 to 2026

How an Outdated PHP Server Gets Compromised

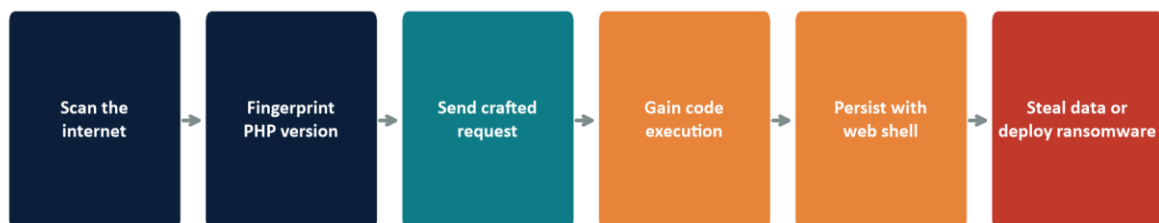


Figure 6.2: Typical six stage compromise chain against an exposed PHP server

Chapter 7. The CMS and Framework Supply Chain Problem

A PHP runtime does not exist in isolation. Almost every commercial or public sector website built in PHP sits on top of a content management system, an ecommerce engine, a framework such as Laravel or Symfony, and dozens of third party packages pulled from Packagist or marketplace repositories. When an organisation lets its PHP runtime fall behind, a chain reaction occurs across the entire software stack.

7.1 How outdated PHP freezes application security updates

Modern PHP frameworks and content management systems move their minimum required PHP version forward every year. Laravel 11 and 12 require PHP 8.2 or higher. Symfony 7 requires PHP 8.2 or higher. Drupal 10 and 11 require PHP 8.1 and 8.3 respectively. Modern Magento and Adobe Commerce releases require PHP 8.2 or 8.3. Even WordPress, which maintains broad backward compatibility, dropped support for PHP 5 and early PHP 7 branches and recommends PHP 8.3 or newer.

When a server stays stuck on PHP 7.4 or PHP 8.0, Composer refuses to install newer releases of security critical libraries such as Guzzle, Symfony HttpFoundation, Monolog, Firebase JWT or league oauth2 server. That means a single outdated runtime blocks two kinds of fixes at once. First, the server misses core PHP engine fixes such as the October 2026 PHP2Shell patches. Second, the application misses library and plugin security updates because the newer patched versions of those libraries require PHP 8.2 or 8.3 syntax and type features.

Software layer	Example component	What happens when PHP stays on 7.4 or 8.0	Security consequence
Core runtime	php fpm, libphp, opcache	Misses upstream engine patches such as CVE 2026 109618 and CVE 2026 109619	Direct unauthenticated remote code execution and worker memory disclosure
Application framework	Laravel, Symfony, Yii, CakePHP	Locked on end of life framework branches that no longer receive security releases	Known deserialization, routing, validation and authentication bypass flaws stay open
Composer packages	guzzlehttp, dompdf, phpmailer, twig	Composer solver pins older minor versions compatible with PHP 7	Server side request forgery, template injection and path traversal bugs remain unpatched

Software layer	Example component	What happens when PHP stays on 7.4 or 8.0	Security consequence
CMS plugins and modules	WordPress plugins, Drupal modules, Magento extensions	Vendors publish security fixes only for current plugin branches requiring PHP 8	Automated botnets exploit known plugin SQL injection and file upload flaws within hours
C library bindings	ext curl, ext openssl, ext libxml, ext gd, imagick	Older bundled PHP builds link against outdated OpenSSL, libxml2 or ImageMagick libraries	TLS downgrade weaknesses, XML external entity parsing and image parsing buffer overflows

Table 7.1: How an outdated PHP version freezes security across five layers of the stack

7.2 WordPress ecosystem metrics

Because WordPress powers such a large share of the web, its public telemetry offers a clear window into administrator habits. Aggregated WordPress ecosystem statistics in late 2026 show approximately 77.4 to 82.6 percent of WordPress sites running on PHP 8 branches, while approximately 12 to 17 percent still operate on PHP 7.4 and under 5 percent remain on versions below PHP 7.4. When we include PHP 8.0 and PHP 8.1, which have both passed their official end of life dates, more than a third of the WordPress ecosystem runs on a PHP version that no longer receives standard upstream security maintenance from the PHP Foundation.

In e-commerce subsets such as WooCommerce stores, approximately one in eight stores continues to process customer carts, billing addresses and payment redirects on PHP 7.4. Store owners hesitate to upgrade because custom shipping calculators, regional tax plugins or payment gateway connectors written five years ago throw fatal errors under PHP 8's stricter type checks.

7.3 Abandonware and orphan code

In nearly every Bithost code audit of a legacy PHP estate, we find at least three dependencies whose original maintainers have archived the repository or stopped responding to security reports. Typical examples include PDF generators such as older HTML2PDF builds, legacy SOAP clients, custom encryption wrappers written around the removed mcrypt extension, and Excel export libraries such as PHPExcel, which was officially deprecated years ago in favour of PhpSpreadsheet.

The removal of the mcrypt extension in PHP 7.2 and the tightening of internal function signatures in PHP 8.0 act as a wall for applications built around those libraries. Rather than rewriting fifty lines of encryption or spreadsheet code to use OpenSSL, Sodium or PhpSpreadsheet, teams keep the whole server on an obsolete PHP version for years.

Chapter 8. Technical Effects of Running Exposed or Outdated PHP

When a website runs an exposed or outdated PHP version, what can an attacker actually do on the system? The answer covers every major category of technical compromise, from silent memory theft to complete takeover of the underlying server and internal corporate network.

8.1 Unauthenticated remote code execution

Remote code execution, abbreviated as RCE, is the most severe technical outcome in web security. In an unauthenticated RCE attack such as CVE 2026 109618, CVE 2026 109619 or CVE 2024 4577, the attacker sends an HTTP request over the public internet without logging in. When the exploit succeeds, the attacker's commands run on the server with the operating system permissions of the PHP worker process, normally `www data`, `apache`, `nginx` or `NT AUTHORITY\SYSTEM` on poorly configured Windows stacks.

Once arbitrary commands run inside the web worker, the attacker does not need to guess passwords or bypass application login screens. The attacker is already inside the house.

8.2 Cross request memory disclosure and state leakage

One of the most alarming aspects of modern PHP engine bugs is what happens inside persistent worker models such as PHP FPM, `mod_php` and FrankenPHP. To achieve high speed, a single PHP worker process handles thousands of requests one after another. When a vulnerability such as CVE 2026 109619 leaves stale global state or unfreed memory pointers inside that worker, or when an out of bounds read occurs in a C extension, data processed during User A's request can still sit in the process heap when the worker picks up Attacker B's request.

- Session identifiers and authentication cookies belonging to logged in administrators or customers.
- Database connection strings, AWS access keys, Stripe or Razorpay secret keys and SMTP passwords loaded from `.env` into memory.
- Plaintext passwords, personal identification numbers, Aadhaar or tax identifiers and credit card details submitted in POST requests moments earlier.
- Internal API bearer tokens and JSON Web Token signing keys used to talk to backend microservices.

8.3 Web shells and hidden persistence

Automated exploitation bots rarely run interactive commands by hand. Their first action after achieving code execution is to establish persistence so they can return even if the administrator later

patches the PHP binary. We regularly uncover four persistence techniques during post breach forensics on PHP servers.

1. Obfuscated PHP web shells dropped into writable upload folders, cache directories or deep inside vendor/composer subdirectories, disguised with filenames such as `class wp http cache.php` or `autoload_static_helper.php`.
2. Modification of `php.ini` or `.user.ini` to set `auto_prepend_file` pointing to a hidden script inside `/tmp` or `/dev/shm`, ensuring malicious code runs on every page load without changing a single visible application file.
3. Creation of rogue administrative accounts directly inside the MySQL, MariaDB or PostgreSQL database table, combined with stealthy hooks in authentication middleware that capture plaintext credentials whenever real administrators log in.
4. User level cron jobs, systemd timers or unauthorized SSH keys added to the web server user's home directory where home folders were mistakenly left writable.

8.4 Payment card skimming and Magecart style injection

On retail stores running Magento, OpenMage, WooCommerce, PrestaShop or custom PHP carts, attackers frequently avoid noisy ransomware altogether. Instead, they inject twenty lines of PHP or JavaScript into the checkout template or payment controller. Whenever a customer enters a payment card number, expiry date and CVV code, the compromised PHP script forwards a copy of the form data to an attacker controlled domain before passing the customer on to the real payment gateway. Because orders continue to succeed normally, store owners often fail to notice a digital skimmer for months.

8.5 Lateral movement into the internal network

A compromised PHP web server is rarely the end of the intrusion. Web servers sit in a privileged network position because they must talk to database servers, Redis or Memcached clusters, internal REST APIs, active directory controllers or shared NFS storage. Once an attacker controls the PHP worker process, they read the local configuration files, extract the database credentials and internal hostnames, and use the web server as a pivot point to attack the rest of the enterprise.

Pivot target	How the PHP server reaches it	What the attacker achieves
MySQL, MariaDB or PostgreSQL database	Credentials stored in plaintext inside <code>.env</code> , <code>wp config.php</code> or <code>app/etc/env.php</code>	Full dump of customer records, order history, password hashes and personal data

Pivot target	How the PHP server reaches it	What the attacker achieves
Redis or Memcached session store	Unauthenticated TCP socket on port 6379 or 11211 inside the VPC	Hijacking of every active user session or remote code execution on the cache host
Cloud metadata service (169.254.169.254)	HTTP requests issued from the compromised PHP process on AWS, Azure or GCP	Theft of temporary IAM role credentials when IMDSv1 is enabled, granting access to S3 buckets and cloud APIs
Internal microservices and ERP APIs	Internal private network routes trusted by firewall rules	Unauthorized financial transactions, invoice tampering or bulk data extraction
Shared file storage (NFS, SMB, EFS)	Mounted media and document directories shared across multiple servers	Planting malware or encrypting files across every server that mounts the same volume

Table 8.1: How attackers pivot from a compromised PHP web server into core enterprise systems

8.6 Cryptojacking, botnet recruitment and SEO poisoning

Not every attacker is looking for corporate secrets. Commodity threat groups scan for outdated PHP servers around the clock and use them for three high volume monetization schemes. First, they launch XMRig or RedTail cryptocurrency miners that pin CPU usage at 100 percent and drive up cloud compute bills. Second, they enroll the server into distributed denial of service botnets or use its clean IP reputation to send phishing emails. Third, they inject hidden links and doorway pages promoting counterfeit pharmaceuticals, illegal gambling or adult content whenever the incoming User Agent string matches Googlebot, destroying the organisation's hard earned search engine rankings.

Chapter 9. Business and Operational Effects on Organisations

For a chief executive, hospital director, university registrar or store owner, buffer overflows and stale worker pointers are abstract engineering concepts. What matters to leadership is what happens to the business on the Monday morning after an unpatched PHP server is compromised. The operational effects follow a painfully consistent pattern.

9.1 Emergency downtime and unplanned maintenance windows

When a flaw as severe as PHP2Shell is disclosed, organisations that have not maintained their PHP estate face an immediate dilemma. Following the 21 October 2026 disclosure and subsequent advisories from national cybersecurity agencies including CISA, CERT EU and the UK NCSC, numerous web infrastructure providers, government portals and enterprises had to restrict access temporarily or initiate emergency maintenance windows while teams scrambled to test patches.

Taking a public portal offline for forty eight hours to perform an emergency jump from PHP 7.4 to PHP 8.4 is far more disruptive than a planned upgrade. Orders stop flowing, citizen applications fail, partner API integrations time out, and support desks are flooded with calls.

9.2 Forensic containment and system rebuilds

If an exposed server was left unpatched during an active exploitation wave, applying the patch three days later does not make the server clean. Any competent incident response team must assume that an automated scanner reached the server during the exposure window. Verifying whether an attacker accessed worker memory or dropped a web shell requires inspecting access logs, auditing file integrity hashes, rotating every secret key and database password that ever touched that server, and rebuilding virtual machines or containers from known clean images.

9.3 Loss of customer trust and brand reputation

Customers forgive a planned two hour maintenance window announced in advance. They rarely forgive receiving a breach notification letter explaining that their home address, phone number, identity documents or payment details were stolen because the company ran software whose support ended four years ago. In competitive sectors such as retail ecommerce, healthcare clinics and financial services, a publicised web breach drives immediate customer churn to competitors.

9.4 Search engine blacklisting and email blocklisting

When commodity attackers compromise an outdated PHP site and inject spam pages or malware droppers, Google Safe Browsing and Microsoft SmartScreen flag the domain within days. Every visitor

who clicks a search result or types the company URL sees a bright red warning screen stating that the site ahead contains harmful programs or deceptive content. At the same time, if the attacker uses the server to send spam, the server's IP address and domain land on Spamhaus and other DNS blocklists, causing legitimate invoices, password resets and staff emails to bounce.

9.5 Performance penalties of staying on legacy PHP

Security is the headline reason to leave old PHP behind, but performance is a close second. Every major release from PHP 7.0 through PHP 8.5 has improved the Zend Engine, OPcache, memory allocation and Just In Time compilation. Independent benchmarks across WordPress, WooCommerce, Laravel and Symfony consistently show that PHP 8.3, 8.4 and 8.5 handle significantly more requests per second with lower memory consumption than PHP 7.4 or PHP 5.6. Organisations running old PHP are literally paying higher monthly cloud hosting bills to run slower, less secure servers.

Chapter 10. Sectors Most Affected by Exposed and Outdated PHP

Every industry uses web applications, yet the risk from outdated PHP is not spread evenly across the economy. To understand where the damage will land hardest, Bithost evaluated ten major sectors across five risk factors: the share of legacy PHP in their public footprint, the sensitivity of the data stored on those servers, how slowly the sector applies runtime patches, the weight of regulatory penalties, and how actively threat groups target those organisations.

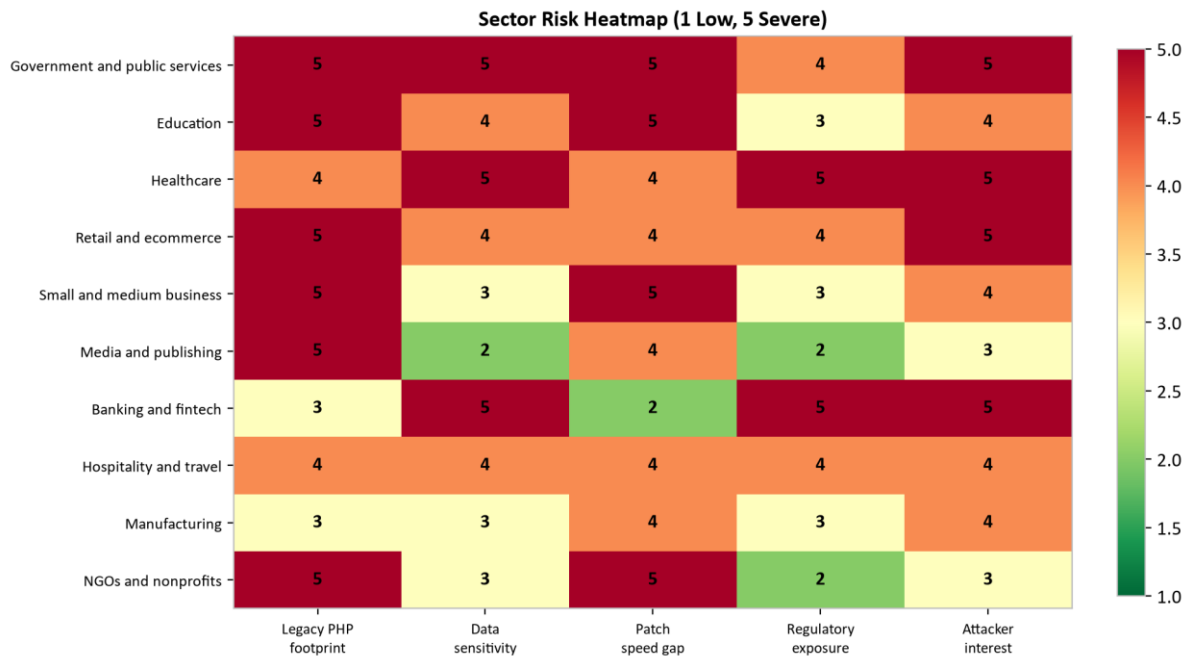


Figure 10.1: Bithost sector risk heatmap across five exposure dimensions

Sector	Overall risk tier	Typical PHP applications exposed	Primary reason upgrades lag
Government and public services	Tier 1 Critical	Citizen service portals, municipal tax sites, recruitment boards, district portals, public grievance systems	Lowest bidder procurement contracts end after launch with no funded maintenance or runtime upgrade budget
Education and universities	Tier 1 Critical	Moodle LMS, student admission portals, exam result portals, departmental Drupal and WordPress sites	Decentralised IT where individual faculties run servers with student admins and only two upgrade windows per year

Sector	Overall risk tier	Typical PHP applications exposed	Primary reason upgrades lag
Healthcare and diagnostics	Tier 1 Critical	Patient appointment portals, diagnostic lab report download sites, clinic websites, billing portals	Fear of breaking connected lab or hospital management interfaces and small IT teams focused on clinical uptime
Retail and ecommerce	Tier 1 Critical	Magento, OpenMage, WooCommerce, PrestaShop, OpenCart stores and dealer ordering portals	Custom checkout plugins, payment connectors and ERP sync scripts break when moving from PHP 7.4 to PHP 8
Small and medium businesses	Tier 1 Critical	Company websites, lead capture forms, customer support portals, simple invoicing systems	No in house security or sysadmin staff, reliance on cheap shared hosting and forgotten agency built code
Media, publishing and news	Tier 2 High	High traffic WordPress and Drupal newsrooms, regional language portals, ad tech trackers	Heavy custom editorial themes and advertising plugins, though larger publishers use CDN shielding
Hospitality, travel and ticketing	Tier 2 High	Hotel booking engines, tour operator portals, regional bus and event ticketing sites	Seasonal business cycles and third party booking widgets pinned to legacy PHP versions
Manufacturing and logistics	Tier 2 High	Supplier portals, dealer extranets, shipment tracking pages, warranty registration portals	IT budgets prioritise factory floor systems while external supplier portals run untouched for a decade
NGOs, charities and associations	Tier 2 High	Donor portals, membership directories, campaign sites on WordPress, Drupal and CiviCRM	Grant funded projects rarely include ongoing infrastructure engineering
Banking, insurance and fintech	Tier 3 Moderate	Marketing sites, partner onboarding portals, legacy subsidiary applications	Core banking is rarely PHP and regulators enforce patching, though outer marketing and partner portals remain exposed

Table 10.1: Sector vulnerability ranking and primary exposure drivers

10.1 Government, municipal and public sector portals

Across national, state and municipal governments, PHP is the workhorse of digital governance. Ministries, district collectorates, police departments, public works divisions, state examination boards and local panchayats or town councils publish thousands of public portals built on Drupal, Joomla, WordPress or custom PHP frameworks. Many of these portals were commissioned under fixed price tenders five to ten years ago. Once the vendor delivered the site and passed the initial acceptance test, the development contract expired.

What happens next is familiar to every public sector auditor. The portal continues to run on a virtual machine in a state data centre or on a leased cloud instance. Nobody owns the job of refactoring old PHP 7.2 or PHP 7.4 code to run on PHP 8.4. When a vulnerability such as PHP2Shell or CVE 2024 4577 breaks, automated scanners hit these government domains within hours. Compromised public portals are used to deface official pages, leak citizen application records containing national ID numbers and addresses, or host malicious files under a trusted government domain extension.

10.2 Education: universities, colleges and school networks

Higher education is one of the most exposed sectors in every internet wide PHP scan. A single large university often operates more than a hundred subdomains. The central admissions website might be maintained by a professional IT team, while individual academic departments, research labs, alumni associations, student clubs, conference committees and library portals run their own WordPress, Drupal, Joomla or Moodle installations on forgotten virtual machines.

Moodle deserves special attention. It is the dominant open source learning management system globally and is written in PHP. Upgrading Moodle to a current release requires upgrading the underlying PHP runtime, the database version and every third party grading or proctoring plugin at the same time. Because universities cannot risk breaking their learning platform in the middle of a semester or during final examinations, upgrades are postponed repeatedly. A breach on a university PHP server exposes student transcripts, identity documents, faculty payroll details and unpublished research.

10.3 Healthcare, hospitals and diagnostic chains

According to the IBM Cost of a Data Breach Report 2025, healthcare remained the costliest industry for data breaches globally for the twelfth consecutive year, with an average breach cost of USD 7.42 million and an average of 279 days to identify and contain an incident. While core electronic health record systems in major hospitals often run on enterprise software stacks, the outer ring of patient interaction is heavily built in PHP.

- Online appointment booking and doctor consultation portals.
- Diagnostic laboratory portals where patients enter a receipt number and download PDF blood test or imaging reports.
- Regional hospital, dental, fertility and specialty clinic websites with patient enquiry forms.
- Third party insurance pre authorisation and medical billing portals.

Diagnostic report portals are a frequent target. Many were written in custom PHP or older Laravel and CodeIgniter versions between 2016 and 2021. If an attacker exploits an unpatched PHP runtime on a lab portal, they gain read access to thousands of downloadable PDF medical reports and the database of patient names, phone numbers, dates of birth and referring doctors.

10.4 Retail, ecommerce and direct to consumer brands

Online retail runs on PHP through WooCommerce, Magento, OpenMage, PrestaShop and OpenCart. For an online merchant, the checkout page is the cash register of the company. Store owners are understandably terrified of touching anything that might lower checkout conversion rates, especially between October and January when fourth quarter holiday and festival sales peak.

That timing makes the late October 2026 PHP2Shell disclosure particularly painful for retail. Merchants entering the peak shopping season of Diwali, Singles Day, Black Friday, Cyber Monday and Christmas face a stark choice: schedule a maintenance window to patch PHP now, or risk having an automated exploit plant a payment card skimmer right as transaction volume hits its annual high.

10.5 Small and medium businesses

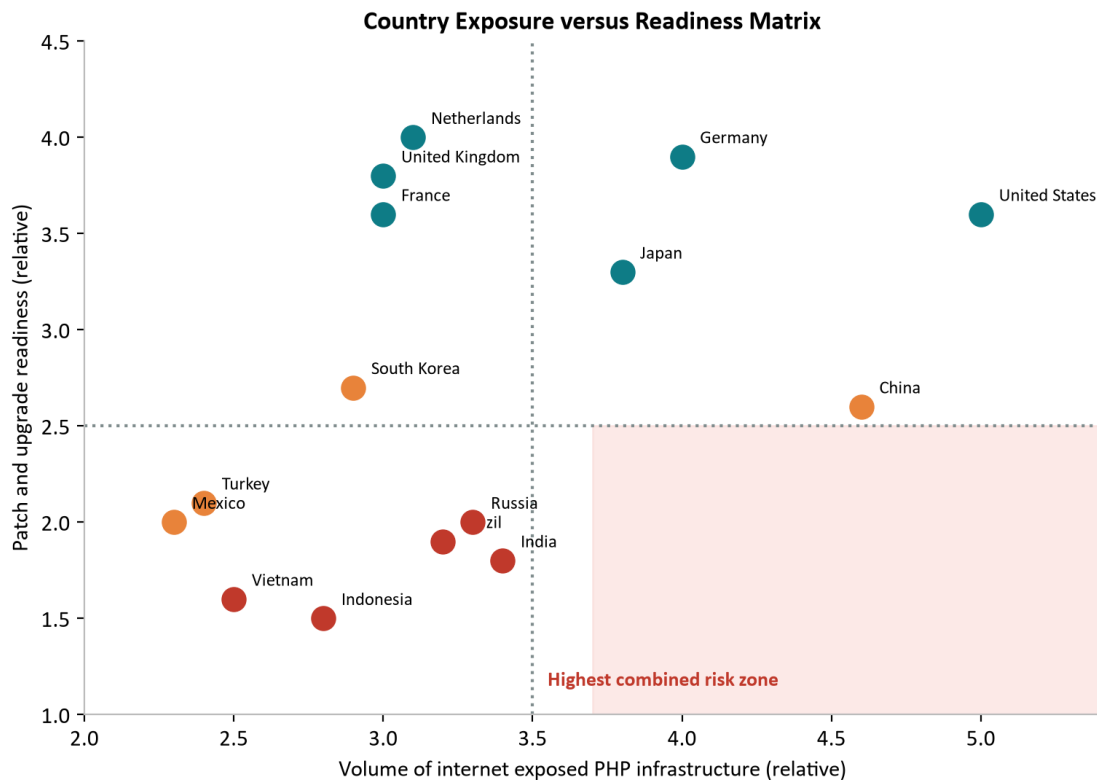
Small and medium businesses make up the largest share of exposed hosts by raw count. A typical fifty person accounting firm, law practice, real estate agency, logistics broker or local manufacturer has a website built on WordPress or custom PHP hosted on a shared server or a basic cloud droplet. The company has no Chief Information Security Officer, no vulnerability scanner and no staging environment. Until the hosting provider forces a PHP version upgrade or the site is visibly defaced, nobody checks what version of PHP is running.

10.6 Banking, financial services, insurance and fintech

Banks and regulated fintech companies rarely run their core ledger on PHP, and central bank supervisors such as the Reserve Bank of India enforce strict vulnerability management timelines. Even so, financial institutions still get caught by outdated PHP through three side doors: marketing and campaign microsites built by external creative agencies, acquired startups that have not yet been migrated onto corporate infrastructure, and third party vendor portals that process loan leads or insurance claims.

Chapter 11. Country and Regional Exposure Analysis

Which country will be affected most by exposed and outdated PHP websites? Answering that question honestly requires looking at two different metrics. The first metric is raw volume: which country hosts the largest physical number of exposed PHP servers inside its data centres? The second metric is combined national impact: which country has a large exposed footprint combined with a high ratio of legacy end of life versions, heavy reliance on PHP across public and commercial services, and slower average patching capacity?



Source: Bithost Research assessment built from public scan reporting, October 2026. Positions are relative.

Figure 11.1: Country exposure volume versus upgrade readiness matrix

11.1 The two lenses: raw hosting volume versus real world vulnerability

When scanners such as Shodan or Censys map exposed PHP servers by IP geolocation, the United States, China and Germany always sit near the top of the chart. That is partly because millions of businesses around the world rent virtual servers in Virginia, Oregon, Frankfurt, Nuremberg, Beijing or Hangzhou from global cloud and hosting companies such as AWS, DigitalOcean, Hetzner, IONOS and Alibaba Cloud. A server physically located in a Frankfurt data centre may actually belong to a retailer in Spain or a startup in Nigeria.

When we look instead at where organisations own and operate high risk legacy PHP applications without dedicated security teams, and where threat intelligence teams record the heaviest concentration of successful compromises, the picture shifts sharply toward fast growing digital economies in Asia and Latin America alongside major Western hosting hubs.

Country or region	Global volume rank (Shodan / scan telemetry)	Dominant risk driver	Bithost combined impact rating
United States	Rank 1 by total exposed hosts	Largest global concentration of cloud instances, WordPress hosting fleets and legacy enterprise apps	Severe (Highest raw volume, highest per breach financial cost at USD 10.22M)
India	Top 10 by exposed hosts	Massive SME, education and municipal PHP footprint, heavy shared hosting usage, DPDPA 2023 penalty exposure	Severe (Highest operational and upgrade readiness gap across SMEs and public portals)
China	Rank 2 by total exposed hosts	Vast domestic web infrastructure with a high proportion of older version revealing PHP deployments	Severe (High volume plus Windows and localized stack exposure)
Germany	Rank 3 to 4 globally	Europe's largest low cost dedicated and VPS hosting hub (Hetzner, IONOS) serving businesses across Europe	High (High server concentration, strict GDPR and NIS2 enforcement)
Japan	Top 5 globally	Long lived custom corporate and ecommerce PHP applications, heavily targeted in CVE 2024 4577 campaigns	High (High legacy custom PHP retention and active regional targeting)
Brazil	Top 10 globally	Rapidly digitized retail, municipal and SME sector with widespread legacy PHP 7 and XAMPP deployments	High (Heavy inbound exploit traffic and ransomware activity)
Indonesia, Vietnam and Malaysia	High regional density	Fast growing ecommerce, university and government portal footprint with limited security staffing	High (High ratio of unpatched end of life PHP installations)
Taiwan and Hong Kong	High per capita target rate	Intense targeting by regional threat groups during PHP CGI and core runtime exploitation waves	High (Consistently targeted within hours of PHP disclosures)

Country or region	Global volume rank (Shodan / scan telemetry)	Dominant risk driver	Bithost combined impact rating
United Kingdom, France and Netherlands	Top 10 hosting hubs	Dense European hosting infrastructure, agency hosted SME estates and public sector Drupal/WordPress sites	Moderate to High (Large footprint balanced by stronger compliance pressure)

Table 11.1: Country and regional risk comparison for exposed and outdated PHP

11.2 United States: highest server count and highest financial loss per breach

The United States ranks first in total internet facing PHP services. That position reflects both the size of the US economy and the fact that major cloud providers and managed WordPress hosts operate huge server fleets inside US regions. While large American enterprises have mature security operations centres, the US also has millions of small businesses, local school districts, municipal governments, medical practices and regional retailers running aging PHP applications. Combined with an average data breach cost of USD 10.22 million reported by IBM in 2025 and aggressive class action litigation, the United States faces the largest total dollar loss from PHP related breaches.

11.3 India: why India faces acute operational impact

In our assessment, India is one of the countries that will feel the operational effects of outdated PHP most acutely over the next twelve to twenty four months. Four factors explain why.

1. Scale of PHP adoption across critical everyday services. From university admission and examination portals to state and district e governance sites, cooperative banks, diagnostic chains, travel portals and millions of MSME websites, PHP is the default language of India's web presence.
2. The agency and freelance delivery model. Hundreds of thousands of Indian businesses commissioned websites from small web development agencies on one time contracts. Those sites were deployed on PHP 5.6, 7.2 or 7.4 on shared cPanel servers or unmonitored cloud instances and have never had a code maintenance budget.
3. Direct exposure to automated scanning. Indian public sector, telecommunications and education networks regularly appear in threat intelligence reports as prime targets for opportunistic web shells, data theft and defacement.
4. A new regulatory landscape. With CERT In's mandatory six hour incident reporting direction and the Digital Personal Data Protection Act 2023 carrying penalties up to INR 250 crore for failing to maintain reasonable security safeguards, running an unpatched end of life runtime that leaks citizen or customer personal data is now a board level legal risk.

11.4 East and Southeast Asia: China, Japan, Taiwan, Singapore, Indonesia and Vietnam

East and Southeast Asia face a dual challenge. First, countries such as China and Japan maintain huge domestic web ecosystems where custom PHP applications built a decade ago remain in daily service across manufacturing, logistics, retail and local government. Second, regional threat actors weaponise new PHP vulnerabilities at extraordinary speed. During the CVE 2024 4577 campaigns, telemetry from Bitdefender and Akamai showed heavy exploitation focused on Taiwan, Hong Kong and Japan, while CISA, CSA Singapore and regional CERTs issued urgent alerts for both CVE 2024 4577 and the October 2026 PHP2Shell flaws. In Indonesia, Vietnam, Malaysia and the Philippines, rapid growth in online retail and campus portals has outpaced the supply of experienced application security engineers.

11.5 Europe: Germany, the Netherlands, the United Kingdom and France

Europe's exposure is concentrated in its hosting backbone. Germany, the Netherlands, France and the United Kingdom host millions of virtual private servers and dedicated machines for small and midsize companies across the continent. While European enterprises have invested heavily in security, the enforcement of the NIS2 Directive alongside GDPR means that European organisations operating vulnerable internet facing infrastructure now face both technical compromise risk and direct supervisory audits of their vulnerability handling and supply chain security practices.

11.6 Latin America, the Middle East and Africa

In Brazil, Mexico, Colombia, the United Arab Emirates, Saudi Arabia, South Africa, Kenya and Nigeria, PHP powers the vast majority of municipal portals, universities, regional news outlets and midsize ecommerce stores. Brazil in particular recorded heavy inbound exploit traffic during recent PHP vulnerability waves, with attackers using exposed web servers to distribute banking trojans, cryptominers and ransomware.

Chapter 12. Deep Dive Sector Scenarios and Failure Patterns

Statistics show the scale of the PHP problem, while concrete operational scenarios show how compromises actually unfold inside real organisations. The ten composite scenarios in this chapter are drawn from patterns observed across real world security assessments, public incident disclosures and emergency upgrade engagements. Each scenario illustrates the technical root cause, the business impact and the exact controls that prevent a repeat.

12.1 Scenario 1: Municipal citizen portal and property tax system

A municipal corporation serving 1.8 million residents launched an online property tax and birth certificate portal in 2018. The site was written in custom PHP 7.1 running on Apache with `mod_php` inside a state data centre virtual machine. The original vendor handed over the source code at the end of a two year maintenance contract in 2020. Between 2020 and 2026, municipal IT staff kept the Linux server running and renewed the SSL certificate each year, yet never upgraded the PHP runtime because the portal appeared to work.

Why it was exposed: The server broadcast `X Powered By: PHP/7.1.33` on every page. The upload module for address proof documents checked only the client supplied Content Type header and stored files inside `/var/www/html/uploads/` without an Nginx or Apache rule blocking script execution.

How compromise unfolds: Automated scanners index the server's banner and hit it with both core runtime exploits and a multipart upload bypass. Once a web shell lands inside the uploads directory, the attacker reads `config/db.php`, connects to the local MySQL instance running as root without a password on `127.0.0.1`, and exports 640,000 citizen records containing names, property IDs, mobile numbers and scanned identity documents.

What prevents it: Migrating the portal to PHP 8.4 under PHP FPM, moving document storage outside the web root, enforcing `open_basedir` and read only source permissions, and running monthly ZScanner checks across all municipal subdomains.

12.2 Scenario 2: Multi campus university running Moodle and departmental WordPress

A public university with 42,000 students operates a central Moodle learning management system along with sixty eight departmental websites, conference microsites and research lab portals. While the central IT team patched the main university homepage, thirty four departmental sites ran on forgotten virtual machines set up by graduate students over the past decade on PHP 5.6, 7.2 and 7.4.

Why it was exposed: Several conference websites shared the same wildcard TLS certificate and internal VLAN as the student records database. Two of those sites ran PHP 7.2 with a forgotten `phpinfo.php` file in the web root that displayed internal environment variables and LDAP bind credentials.

How compromise unfolds: An attacker discovers the `phpinfo.php` page through a search engine query, extracts the LDAP credentials, and uses an unauthenticated PHP runtime exploit to gain a shell inside the campus DMZ. From that foothold, the attacker pivots into the internal network during exam week and deploys ransomware across shared departmental file servers.

What prevents it: Centralising all departmental sites onto a managed multi tenant platform or container cluster running patched PHP 8.4/8.5, isolating legacy research servers onto an untrusted VLAN, removing diagnostic files, and running Bithost CodeSecurity across all campus Git repositories.

12.3 Scenario 3: Regional diagnostic laboratory chain report portal

A diagnostic chain with ninety collection centres provides patients with an online portal to download blood test and radiology PDF reports. The portal was built on Laravel 5.8 and PHP 7.3 in 2019. Because the laboratory information system pushed reports to the web server over a custom SFTP script every five minutes, the IT manager feared that upgrading PHP or Laravel would disrupt report delivery.

Why it was exposed: Being locked on PHP 7.3 prevented the team from upgrading Laravel and its dependencies. Both the PHP runtime and the application framework carried known remote code execution flaws.

How compromise unfolds: An opportunistic bot exploits the unpatched runtime, dumps the patient table and scraped PDF reports from `/var/www/html/storage/reports`, and posts a sample of 50,000 patient diagnostic records on an extortion forum.

What prevents it: Using Rector and Bithost CodeSecurity to upgrade the Laravel application to a supported release on PHP 8.4, storing patient PDFs on encrypted object storage with short lived signed download URLs, and auditing the server configuration prior to DPDPA 2023 readiness certification.

12.4 Scenario 4: Direct to consumer WooCommerce and Magento retail brand

A fast growing apparel brand generating INR 85 crore in annual online sales runs a heavily customised store with forty two third party plugins for loyalty points, WhatsApp cart recovery, regional courier tracking and flash sales. Three of those plugins were customized directly inside the `wp content/plugins` folder by a former freelancer, blocking updates since 2022 and keeping the production server pinned to PHP 7.4.

Why it was exposed: During peak pre festival sales in late October, the store ran unpatched PHP 7.4 alongside outdated checkout plugins on a single large cloud instance.

How compromise unfolds: Attackers exploit the unpatched stack to inject a twenty line JavaScript and PHP skimmer into the checkout template. For nineteen days, every customer who enters card or billing details has their information copied to an external domain before payment redirection. Customer complaints to issuing banks trigger a mandatory forensic investigation, temporary suspension of merchant card processing and lakhs in emergency remediation costs.

What prevents it: Extracting custom plugin edits into clean child hooks compatible with PHP 8.4, running file integrity monitoring on all checkout templates, enforcing strict Content Security Policy headers and scanning the live checkout flow weekly with ZScanner.

12.5 Scenario 5: Manufacturing supplier extranet on Windows XAMPP

An automotive components manufacturer runs a supplier invoice and purchase order portal on a Windows Server machine in its head office DMZ. Back in 2017 an internal developer installed XAMPP on Windows to get Apache, PHP and MySQL running quickly, and exposed port 443 through the corporate firewall.

Why it was exposed: XAMPP on Windows enabled PHP CGI mappings alongside an outdated PHP 7.x binary running under a high privilege local Windows account, with SMB file sharing open to the internal corporate LAN.

How compromise unfolds: Attackers exploit PHP CGI argument injection (CVE 2024 4577) or core runtime memory corruption (CVE 2026 109618), gain code execution as the Windows service account, dump cached domain credentials from memory, and move laterally to encrypt the ERP and production scheduling servers.

What prevents it: Auditing Windows hosts with Bithost EWSAF, replacing XAMPP with a hardened IIS FastCGI or Linux PHP FPM deployment running PHP 8.4.27 or 8.5.12, and segmenting supplier portals away from internal Active Directory networks.

12.6 Scenario 6: Digital marketing agency managing 120 client websites

A midsize creative agency hosts 120 client websites on three large cPanel and LiteSpeed servers. Most client sites were built for three month product launches and then left on the server under small annual hosting retainers. All 120 sites ran in default PHP pools, and forty of them were still set to PHP 7.3 or 7.4 in MultiPHP Manager.

Why it was exposed: One abandoned microsite for a 2021 real estate project sat on an unpatched PHP runtime and shared a database server with active enterprise clients.

How compromise unfolds: A bot compromises the forgotten 2021 microsite, uses a local privilege or misconfigured pool permission to browse neighboring directories, and defaces thirty client websites on the same morning.

What prevents it: Enforcing per account CageFS or dedicated FPM user isolation, retiring ended campaign microsites into static HTML archives, and setting a policy that no client site on agency infrastructure may run an end of life PHP branch.

Chapter 13. Detailed Country and Regional Risk Profiles

Chapter 11 introduced our global matrix comparing raw hosting exposure against national upgrade readiness. This chapter examines the specific structural, hosting and regulatory dynamics across eight major countries and regions so multinational security teams and regional leaders can tailor their response.

13.1 India: Digital public scale meets MSME technical debt

India's digital economy has expanded at extraordinary speed over the last decade. Alongside modern cloud native platforms, millions of small and medium enterprises, educational institutions, hospitals, cooperative societies, local bodies and regional retailers rely on PHP websites hosted across domestic and international data centres in Mumbai, Chennai, Hyderabad, Noida, Pune and Bengaluru.

Indian subsector	Typical hosting model	Dominant PHP versions seen in field audits	Key compliance and business risk
State, district and municipal portals	State data centres, NIC cloud or domestic VPS providers	PHP 7.2, 7.4, 8.0, 8.1	CERT In 6 hour reporting mandate, public service disruption, citizen data leakage
Universities, engineering colleges and schools	On premise Linux/Windows servers or budget shared hosting	PHP 5.6, 7.3, 7.4, 8.1	Student identity and exam record leaks, campus ransomware pivots
Hospitals, clinics and diagnostic labs	Shared hosting or single cloud VMs managed by third party vendors	PHP 7.3, 7.4, 8.0	DPDPA 2023 penalties up to INR 250 crore for sensitive health data breaches
D2C brands and regional ecommerce	Managed WordPress, cloud VPS or cPanel servers	PHP 7.4, 8.1, 8.2	Checkout skimming, payment gateway suspension, festival sale downtime
MSMEs, exporters and professional services	Low cost shared cPanel/LiteSpeed hosting	PHP 5.6, 7.2, 7.4	Email blacklisting, SEO poisoning, wire transfer invoice fraud

Table 13.1: Breakdown of PHP exposure across Indian industry subsectors

For Indian organisations, the convergence of the October 2026 PHP2Shell vulnerabilities, the December 2026 PHP 8.2 end of life cliff, CERT In's strict six hour incident reporting rules and the enforcement architecture of the Digital Personal Data Protection Act 2023 makes a structured PHP upgrade and vulnerability scanning programme one of the highest return security investments available in 2026 and 2027.

13.2 United States and Canada: Cloud concentration and litigation risk

North America accounts for the largest absolute count of internet reachable PHP hosts in global scan datasets. While Fortune 500 engineering teams increasingly run containerised workloads behind enterprise WAFs, North America also contains hundreds of thousands of regional healthcare providers, community colleges, county governments, credit unions, franchises and online merchants running legacy WordPress, Drupal, Magento and custom LAMP stacks. In the United States, state data breach notification laws, Federal Trade Commission enforcement against unfair or deceptive security practices, HIPAA rules for healthcare entities and class action lawsuits pushed the average breach cost to USD 10.22 million in 2025.

13.3 European Union and United Kingdom: Hosting hubs under NIS2 and GDPR

Germany, the Netherlands, France and the United Kingdom serve as the physical hosting backbone for Europe. Because providers such as Hetzner, OVHcloud and IONOS offer high performance dedicated and virtual servers at competitive prices, businesses across the entire European Union deploy PHP workloads in those data centres. Under GDPR Article 32 and the NIS2 Directive, European supervisory authorities treat timely patching and lifecycle management of internet facing runtimes as a baseline legal obligation.

13.4 East Asia: Japan, Taiwan, Hong Kong and South Korea

East Asian networks have experienced some of the earliest and most intense exploitation waves whenever a critical PHP vulnerability is published. During the CVE 2024 4577 campaigns, telemetry from multiple security vendors recorded more than half of early localized detections targeting Taiwan and Hong Kong, alongside heavy probing of Japanese technology, telecom and ecommerce servers. Many midsize enterprises in Japan and South Korea run custom built PHP business portals written ten to fifteen years ago that require careful character encoding and database migration planning during upgrades to PHP 8.4 or 8.5.

13.5 Southeast Asia: Singapore, Indonesia, Vietnam, Malaysia and the Philippines

Southeast Asia combines high mobile internet penetration with a booming digital commerce sector built largely on PHP content management systems and frameworks. Singapore's Cyber Security Agency (CSA) regularly publishes rapid advisories on critical PHP vulnerabilities, including both CVE 2024 4577 and the October 2026 PHP2Shell flaws, helping large enterprises patch quickly. Across Indonesia, Vietnam, Malaysia and the Philippines, thousands of fast growing retailers, universities and public agencies rely on small development teams that need automated scanning and turnkey upgrade assistance.

13.6 Middle East, Africa and Latin America

Across the Gulf Cooperation Council states, South Africa, Nigeria, Kenya, Brazil, Mexico and Colombia, digital transformation across government ministries, tourism boards, universities and retail groups has relied heavily on PHP platforms. In Brazil and Mexico, threat actors actively scan for unpatched PHP servers to host regional phishing kits and banking malware payloads, making perimeter discovery and runtime upgrades essential.

Chapter 14. Thirty Country PHP Exposure and Readiness

Reference Directory

To support multinational organisations, hosting providers and public sector leaders operating across jurisdictions, Table 27.1 compiles a thirty country reference profile covering national cybersecurity authority advisories, dominant PHP hosting patterns, primary exposed sectors and statutory breach notification windows.

Country	National CERT / Authority	Primary PHP exposure pattern	Most affected domestic sectors	Breach reporting window & key law
India	CERT In (cert.in.org.in)	High share of SME shared hosting, campus portals and legacy e governance sites	Education, municipal bodies, healthcare labs, D2C retail, MSMEs	6 hours to CERT In + DPDPA 2023 (up to INR 250 crore penalty)
United States	CISA (cisa.gov)	Largest raw count of cloud instances, WordPress fleets and regional business portals	Healthcare, local government, K 12 & higher ed, retail ecommerce	SEC 4 business days (public cos), CIRCIA 72h, state laws & HIPAA
China	CNCERT / NVDB	Large domestic web footprint with high retention of older PHP 5/7 and Windows stacks	Manufacturing, domestic ecommerce, municipal portals, telecom	Immediate / 1 to 24 hours under Cybersecurity Law & Data Security Law
Germany	BSI (bsi.bund.de)	Major European dedicated/VPS hosting hub (Hetzner, IONOS) serving EU SMEs	Hosting providers, midsize manufacturing (Mittelstand), retail, agencies	72 hours under GDPR + 24h early warning under NIS2
Japan	JPCERT/CC (jpcert.or.jp)	Long lived custom corporate PHP portals and Windows localized deployments	Technology, telecom, ecommerce, regional municipalities	Prompt / 3 to 5 days initial report under APPI
United Kingdom	NCSC UK (ncsc.gov.uk)	Dense agency hosted WordPress/Drupal estate across councils, charities and retail	Local councils, universities, NHS partner sites, retail, charities	72 hours to ICO under UK GDPR
Brazil	CERT.br (cert.br)	High volume of legacy PHP 7 and	Retail, municipal portals, universities,	Reasonable timeframe (typically

Country	National CERT / Authority	Primary PHP exposure pattern	Most affected domestic sectors	Breach reporting window & key law
		XAMPP servers targeted by ransomware and banking bots	regional clinics, logistics	3 business days) to ANPD under LGPD
France	ANSSI / CERT FR	Large cloud and dedicated hosting footprint (OVHcloud) plus public Drupal/Symfony sites	Public sector, retail, media, hospitality, higher education	72 hours to CNIL under GDPR + NIS2
Netherlands	NCSC NL	High density European data centre and cloud transit hub hosting international PHP sites	Hosting & cloud providers, ecommerce, logistics, digital agencies	72 hours to AP under GDPR + NIS2
Indonesia	BSSN / ID SIRTII	Rapidly expanding ecommerce, university and local government PHP footprint	Universities, regional government (pemda), online retail, clinics	3 x 24 hours (72 hours) under Personal Data Protection Law (UU PDP)
Singapore	CSA SingCERT	Regional cloud hub with proactive SingCERT advisories on CVE 2024 4577 & PHP2Shell	SME retail, professional services, hospitality, regional portals	As soon as practicable (within 3 calendar days) to PDPC under PDPA
Taiwan	TWCERT/CC	Heavily targeted within hours of PHP CGI and core runtime disclosures	Manufacturing supply chain, technology SMEs, universities, ecommerce	72 hours under Cyber Security Management Act & PDPA
South Korea	KrCERT/CC (KISA)	High volume of legacy PHP 7.x and 8.0 commercial sites and web boards	SME ecommerce, education academies, hospitality, media	Within 24 to 72 hours to KISA / PIPC under PIPA
Vietnam	VNCERT/CC	Fast growing SME and campus web estate with high ratio of unpatched PHP 7	Education, retail, tourism, municipal portals, manufacturing	72 hours after detection under Decree 13/2023/ND CP
Malaysia	MyCERT (CyberSecurity Malaysia)	Widespread WordPress, Joomla and Laravel usage	Higher education, state agencies, retail, healthcare clinics	72 hours under amended Personal Data Protection Act

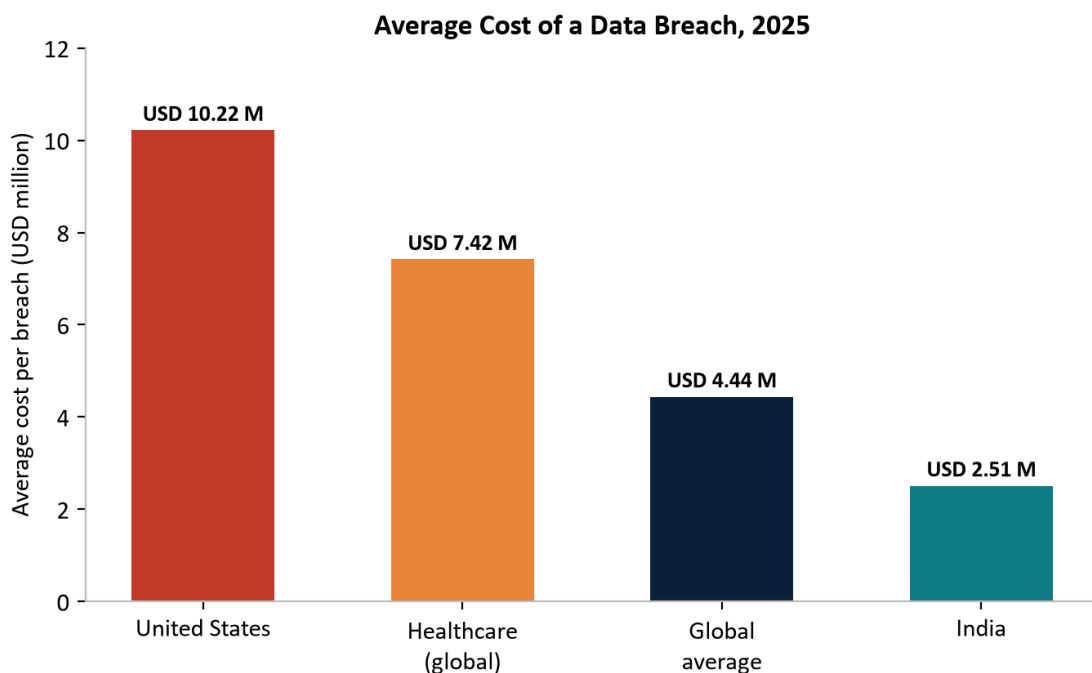
Country	National CERT / Authority	Primary PHP exposure pattern	Most affected domestic sectors	Breach reporting window & key law
		across SMEs and public agencies		
Australia	ASD ACSC (cyber.gov.au)	Large WordPress and WooCommerce adoption across small businesses and councils	Small business, healthcare clinics, education, local councils, retail	30 days assessment / immediate for critical infrastructure (SOI Act)
Canada	Canadian Centre for Cyber Security	Broad public sector Drupal/WordPress usage and SME cloud hosting	Municipalities, education, healthcare clinics, retail, nonprofits	As soon as feasible under PIPEDA / Quebec Law 25
Spain	INCIBE CERT / CCN CERT	High adoption of WordPress, PrestaShop and Moodle across businesses and universities	Universities, tourism & hospitality, retail, municipal councils	72 hours to AEPD under GDPR + NIS2
Italy	CSIRT Italia (ACN)	Extensive SME manufacturing, tourism and municipal PHP website footprint	Tourism, fashion/retail SMEs, municipalities, healthcare	72 hours to Garante under GDPR + NIS2
Poland	CERT Polska (NASK)	Fast growing central European hosting and ecommerce sector using PHP frameworks	Ecommerce, logistics, universities, local government	72 hours to UODO under GDPR + NIS2
Turkey	USOM (TR CERT)	Large domestic hosting ecosystem serving regional retail, media and manufacturing	Retail, hospitality, manufacturing SMEs, universities	72 hours to KVKK Board under Personal Data Protection Law
Mexico	CERT MX	Widespread municipal, university and SME websites on legacy PHP 7 stacks	Municipal portals, education, retail, hospitality, logistics	Without delay to affected data subjects under LFPDPPP
United Arab Emirates	aeCERT (TDRA)	High speed digital commerce, hospitality and real estate portals on WordPress/Laravel	Real estate, hospitality, retail, private clinics, trading SMEs	Immediate notification to UAE Data Office under Federal Decree Law 45

Country	National CERT / Authority	Primary PHP exposure pattern	Most affected domestic sectors	Breach reporting window & key law
Saudi Arabia	Saudi CERT (NCA)	Expanding enterprise and government portal ecosystem under strict NCA ECC controls	Retail, construction/logistics suppliers, education, private healthcare	72 hours to SDAIA under PDPL
South Africa	CSIRT SA / Cybersecurity Hub	Continent's largest hosting hub serving Southern African SMEs, schools and retailers	SMEs, universities, municipal bodies, retail, NGOs	As soon as reasonably practicable to Information Regulator under POPIA
Nigeria	ngCERT	Fast growing fintech peripheral portals, universities, media and ecommerce on PHP	Universities, media, ecommerce, public agencies, SMEs	72 hours to NDPC under Nigeria Data Protection Act 2023
Kenya	KE CIRT/CC	East African hub for education, NGO, SACCO and SME web applications	Education, NGOs, cooperative societies, tourism, retail	72 hours to ODPC under Data Protection Act
Philippines	NCERT (DICT)	Heavy usage of WordPress, Joomla and custom PHP across schools, LGUs and retail	Local government units, universities, healthcare, retail	72 hours to National Privacy Commission under Data Privacy Act
Thailand	ThaiCERT (NCSA)	Strong tourism, hospitality, hospital and retail web presence built on PHP	Hospitality, private hospitals, ecommerce, universities	72 hours to PDPC under PDPA
Colombia	ColCERT	Growing municipal, education and SME ecommerce footprint on shared Linux/Windows hosts	Municipalities, education, retail, logistics SMEs	15 business days to SIC under Law 1581

Table 14.1: Thirty country reference directory of PHP exposure patterns and regulatory reporting duties

Chapter 15. What Organisations Stand to Lose: Financial and Strategic Impact

Keeping an outdated PHP server online often looks free on paper. There is no software licence invoice to approve and no developer bill to pay this month. That apparent saving is an accounting illusion. When an unpatched runtime is exploited, the losses arrive all at once and dwarf the cost of a planned upgrade by fifty to five hundred times.



Source: IBM and Ponemon Institute, Cost of a Data Breach Report 2025. India figure INR 220 million.

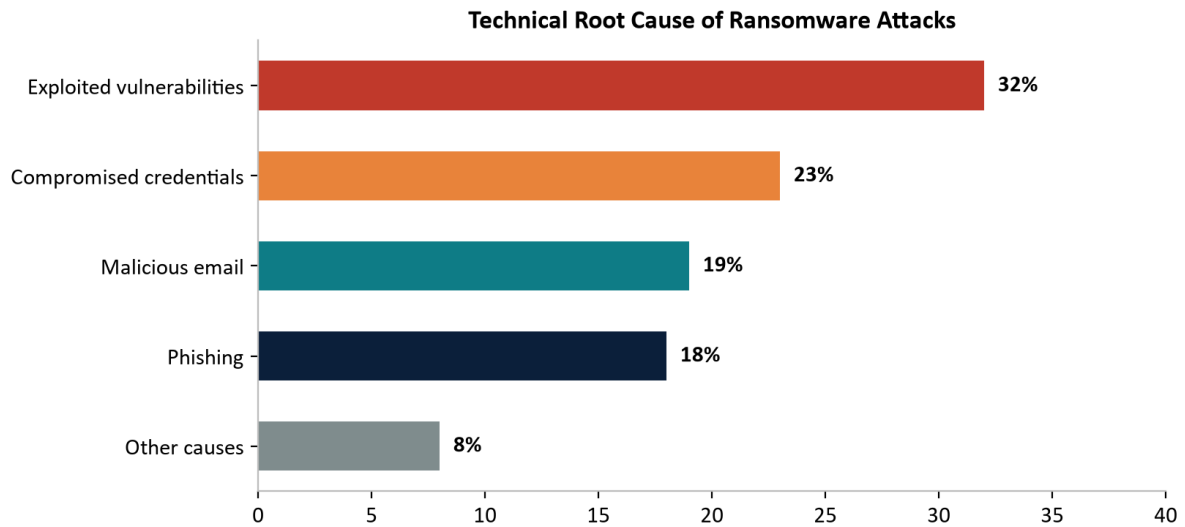
Figure 15.1: Average cost of a data breach in 2025, IBM and Ponemon Institute

15.1 Verified global breach and ransomware cost benchmarks

According to the IBM and Ponemon Institute Cost of a Data Breach Report 2025, the global average cost of a data breach stands at USD 4.44 million. In the United States the average reached a record USD 10.22 million. In India the average cost reached an all time high of INR 220 million, approximately USD 2.51 million. In the healthcare sector globally, the average breach cost stood at USD 7.42 million, with breaches taking an average of 279 days to identify and contain.

Ransomware adds a second set of hard numbers. According to Sophos in its State of Ransomware 2025 survey of 3,400 IT and cybersecurity leaders, exploited vulnerabilities ranked as the number one technical root cause of ransomware attacks for the third consecutive year, accounting for 32 percent of all incidents, ahead of compromised credentials at 23 percent. The mean cost to recover from a ransomware attack, excluding any ransom paid, was USD 1.53 million. Meanwhile, the 2025 Verizon

Data Breach Investigations Report recorded vulnerability exploitation as the initial access vector in 20 percent of all breaches analysed, a 34 percent year over year increase.



Source: Sophos, The State of Ransomware 2025 (3,400 respondents)

Figure 15.2: Technical root causes of ransomware attacks, Sophos 2025

15.2 The seven buckets of loss in a PHP server breach

To help leadership teams budget realistically, the table below breaks down where the money actually goes when a public facing PHP application is compromised.

Loss category	What triggers the cost	Typical range for SME (INR / USD)	Typical range for midsize or enterprise
1. Emergency incident response and forensics	Hiring external responders, log analysis, malware reverse engineering, rebuilding servers from clean images	INR 5 lakh to 25 lakh (USD 6K to 30K)	INR 50 lakh to 4 crore (USD 60K to 500K)
2. Revenue lost during downtime	Storefront, booking engine or API offline during emergency patching or ransomware recovery	INR 2 lakh to 50 lakh (USD 2.5K to 60K)	INR 1 crore to 25 crore (USD 120K to 3M)
3. Customer notification, support and credit monitoring	Mandatory breach notices, call centre surge, legal review of communications	INR 1 lakh to 15 lakh (USD 1.2K to 18K)	INR 30 lakh to 5 crore (USD 35K to 600K)
4. Payment card fines and PCI DSS assessments	Forensic investigation by a PCI Forensic Investigator after checkout skimming, card reissuance fees, higher transaction tiers	INR 10 lakh to 75 lakh (USD 12K to 90K)	INR 1 crore to 15 crore (USD 120K to 1.8M)

Loss category	What triggers the cost	Typical range for SME (INR / USD)	Typical range for midsize or enterprise
5. Regulatory fines and legal defence	Proceedings under DPDPA 2023, GDPR, sector regulators or customer lawsuits	Statutory exposure up to INR 250 crore under DPDPA	Multi million USD or EUR under GDPR/NIS2 or up to INR 250 crore in India
6. Emergency code rewrite under fire	Paying contractors overtime rates to migrate broken PHP 5 or PHP 7 code to PHP 8.4 while systems are down	INR 3 lakh to 20 lakh (USD 3.5K to 25K)	INR 25 lakh to 3 crore (USD 30K to 360K)
7. Long term customer churn and SEO loss	Customers switching to competitors, loss of organic search traffic after Google Safe Browsing warnings	10 to 30 percent drop in quarterly web leads	Multi year brand discounting and higher customer acquisition cost

Table 15.1: Bithost breakdown of direct and indirect losses following a PHP web compromise

15.3 The cost of planned upgrade versus the cost of breach

Compare the figures in Table 12.1 with the cost of proactive prevention. Running a full vulnerability scan, auditing your codebase for PHP 8.4 or 8.5 compatibility, refactoring deprecated functions and moving a typical web application onto a supported, hardened PHP runtime costs a tiny fraction of a single incident response retainer. Even more importantly, a planned upgrade happens on your schedule during calm working hours rather than at two in the morning while an attacker is inside your server.

Chapter 16. Regulatory, Compliance and Cyber Insurance

Exposure

Ten years ago, running an end of life programming language on a web server was seen as a purely technical decision. In 2026, regulators, payment networks and cyber insurance underwriters treat known, unpatched end of life software on an internet facing system as a failure of basic duty of care.

16.1 India: DPDPA 2023 and CERT In directions

Organisations operating in India or processing digital personal data of individuals in India face two direct legal obligations.

- Digital Personal Data Protection Act 2023 (DPDPA). Section 8(5) requires every Data Fiduciary to protect personal data in its possession or under its control by taking reasonable security safeguards to prevent a personal data breach. Under the Schedule to the Act, a breach in observing the obligation of a Data Fiduciary to take reasonable security safeguards to prevent a personal data breach carries a financial penalty of up to INR 250 crore, and failure to notify the Data Protection Board and affected individuals of a breach carries a penalty of up to INR 200 crore. Defending a claim that a server running PHP 7.2 or an unpatched PHP2Shell runtime constituted reasonable security safeguards is extraordinarily difficult.
- CERT In Cyber Security Directions of April 2022. Every service provider, intermediary, data centre, body corporate and government organisation must report specified cybersecurity incidents, including unauthorized access to IT systems/data and exploitation of vulnerabilities, to CERT In within 6 hours of noticing such incidents or being brought to notice about such incidents. Organisations must also synchronise system clocks via NTP and retain ICT system logs securely for a rolling period of 180 days within the Indian jurisdiction.
- Sectoral mandates from RBI, SEBI and IRDAI. Regulated financial entities must conduct periodic vulnerability assessments and penetration tests, patch critical vulnerabilities on internet facing systems within strict timelines, and maintain an accurate inventory of software assets and lifecycle dates.

16.2 European Union and United Kingdom: GDPR, NIS2 and Cyber Resilience

Under Article 32 of the EU and UK General Data Protection Regulation, controllers and processors must implement appropriate technical and organisational measures to ensure a level of security appropriate to the risk. European data protection authorities have repeatedly fined organisations where personal data was stolen through a known, unpatched web vulnerability or an end of life software component, ruling that running unsupported software on an internet facing server violates Article 32. Administrative fines can reach EUR 20 million or 4 percent of global annual turnover. Under

the NIS2 Directive, essential and important entities across energy, transport, health, digital infrastructure, public administration and manufacturing face mandatory vulnerability handling duties and personal management liability.

16.3 PCI DSS 4.0 for any site touching payment flows

Under Requirement 6.3 and 6.4 of Payment Card Industry Data Security Standard (PCI DSS) 4.0, merchants and service providers must identify security vulnerabilities, assign risk rankings, install critical security patches within one month of release, and protect public facing web applications against known attacks. Running an end of life PHP version that no longer receives security patches automatically causes a failure during a Qualified Security Assessor audit or an Approved Scanning Vendor external scan.

16.4 Cyber insurance warranty exclusions

Cyber insurance proposal forms now ask explicit questions about end of life software and patch management timelines for internet facing assets. Many policies include warranties or exclusions stating that the insurer may reduce or deny coverage if a loss arises from a failure to apply critical security patches within a specified window or from running unsupported software without documented compensating controls.

Chapter 17. Regulatory and Security Standard Compliance

Mapping

Security and compliance officers regularly need to translate technical PHP findings into the exact clauses of national laws, sector regulations and international frameworks. Table 26.1 maps the risks of outdated PHP and the corresponding Bithost controls directly to ten major standards.

Framework or law	Specific clause or requirement	How outdated or exposed PHP violates it	How Bithost tools and upgrade services satisfy it
India DPDPA 2023	Section 8(5) Reasonable Security Safeguards and Section 8(6) Personal Data Breach Notification	Running end of life PHP or unpatched PHP2Shell exposes citizen and customer personal data to trivial unauthenticated extraction, carrying penalties up to INR 250 crore.	Turnkey upgrade to PHP 8.4/8.5, ZScanner verification, CodeSecurity SAST/SCA and DPDPA readiness certification via cert.bithost.in.
India CERT In Directions (Apr 2022)	Mandatory 6 hour reporting of vulnerability exploitation / unauthorized access, 180 day local log retention and NTP sync	Unmonitored PHP servers lack centralized error/access logging, making 6 hour forensic root cause reporting impossible.	Hardened PHP FPM logging templates, NTP and 180 day log retention checks, and EWSAF Windows log auditing.
RBI Cyber Security Framework & Master Directions	Baseline cyber security controls, timely patching of critical vulnerabilities on internet facing systems, VAPT and software inventory	Outdated PHP on banking partner portals or NBFC applications fails mandatory periodic VAPT and patch window audits.	ZScanner authenticated DAST, CodeSecurity CycloneDX SBOM generation and rapid patch sprint delivery.
PCI DSS v4.0	Req 6.2 (Bespoke and custom software security), Req 6.3 (Vulnerability identification & 30 day critical patching), Req 6.4 (Public web app protection)	End of life PHP 5/7/8.0/8.1 cannot meet Req 6.3 because upstream critical patches no longer exist. Unprotected checkouts risk Magecart skimming.	ZScanner native PCI DSS 4.0 compliance mapping, CodeSecurity SAST, WAF rule deployment and runtime upgrade to supported PHP 8.4/8.5.
ISO/IEC 27001:2022	Annex A.8.8 (Management of technical vulnerabilities), A.8.9 (Configuration management), A.8.28 (Secure coding)	Lack of PHP version tracking, default expose_php=On configs and unreviewed legacy code trigger major nonconformities during surveillance audits.	Legac o Meter asset scoring, hardened php.ini baselines, CodeSecurity secure coding checks and ISO 27001 readiness via cert.bithost.in.

Framework or law	Specific clause or requirement	How outdated or exposed PHP violates it	How Bithost tools and upgrade services satisfy it
OWASP Top 10 (2021)	A06:2021 Vulnerable and Outdated Components, A05:2021 Security Misconfiguration, A03:2021 Injection	Running end of life PHP is the textbook example of A06:2021, while exposed headers and phpinfo files violate A05:2021.	ZScanner maps all 19 scan modules directly to OWASP Top 10 2021 and OWASP API Security Top 10 2023.
EU & UK GDPR	Article 32 (Security of processing: state of the art technical measures) and Article 33 (72 hour breach notification)	European regulators consistently rule that running unsupported end of life software on internet facing systems breaches Article 32.	Runtime modernisation, encryption upgrades from mcrypt to libsodium, and GDPR certification via cert.bithost.in.
EU NIS2 Directive	Article 21 (Cybersecurity risk management measures, supply chain security and coordinated vulnerability disclosure)	Essential and important entities using vulnerable PHP portals or unvetted Composer supply chains face administrative fines and management liability.	CodeSecurity Composer SCA + CycloneDX SBOMs combined with continuous ZScanner perimeter validation.
US HIPAA Security Rule	45 CFR 164.308(a)(1) Security Management Process and 164.312 Technical Safeguards for ePHI	Healthcare appointment and lab portals on legacy PHP expose electronic protected health information to unauthenticated RCE.	EWSAF HIPAA module for Windows servers, ZScanner offline scanning (zero ePHI leaves the network) and PHP 8.4 migration.
SOC 2 (AICPA Trust Services Criteria)	CC6.1 (Logical access security), CC6.8 (Malicious software prevention), CC7.1 (Vulnerability detection and configuration monitoring)	Auditors flag unpatched internet facing PHP versions and missing vulnerability scans as exceptions in Type II reports.	Monthly ZScanner reports, CodeSecurity CI/CD gates and EWSAF SOC 2 audit evidence.

Table 17.1: Mapping PHP exposure and Bithost controls to ten regulatory and compliance frameworks

Chapter 18. Pre Measures and Immediate Emergency Playbook

Before an organisation can complete a full migration to PHP 8.4 or PHP 8.5, it needs pre measures that stop bleeding immediately. Whether you are responding to the October 2026 PHP2Shell advisories today or preparing your estate against the next zero day disclosure, follow the step by step playbook in this chapter.

18.1 Phase 1: Emergency discovery in the first 24 hours

You cannot protect a PHP server you do not know exists. Shadow IT, forgotten marketing microsites, old staging domains and Docker containers inside developer virtual machines are the most common blind spots. Run the commands below across your infrastructure and perimeter.

```
# 1. Check CLI and FPM binary versions on every Linux host
php -v
php-fpm -v || php-fpm8.4 -v || php-fpm8.3 -v || php-fpm8.2 -v

# 2. Find every php binary installed on the filesystem (catches MultiPHP and custom builds)
find / -name 'php' -o -name 'php-fpm*' -o -name 'libphp*.so' 2>/dev/null

# 3. Check running PHP worker processes and when they were started
ps -eo pid,user,lstart,cmd | grep -E 'php|apache2|httpd|lsphp|frankenphp'

# 4. Inspect running Docker containers for embedded PHP runtimes
docker ps --format '{{.ID}} {{.Names}} {{.Image}}' | while read id name img; do
  ver=$(docker exec "$id" php -r 'echo PHP_VERSION;' 2>/dev/null)
  [ -n "$ver" ] && echo "Container $name ($img) runs PHP $ver"
done
```

18.2 Phase 2: Immediate patching for supported branches (PHP 8.2, 8.3, 8.4, 8.5)

As stated in the official PHP advisory of 21 October 2026, updating is the only complete solution for CVE 2026 109618 and CVE 2026 109619. Ensure every host reaches at least the minimum safe versions below.

Active PHP branch	Minimum patched version for PHP2Shell	Verification command after worker restart
PHP 8.5	8.5.12 or later	php r 'echo PHP_VERSION,' && systemctl restart php8.5 fpm
PHP 8.4	8.4.27 or later	php r 'echo PHP_VERSION,' && systemctl restart php8.4 fpm
PHP 8.3	8.3.36 or later	php r 'echo PHP_VERSION,' && systemctl restart php8.3 fpm

Active PHP branch	Minimum patched version for PHP2Shell	Verification command after worker restart
PHP 8.2	8.2.35 or later (Plan migration before 31 Dec 2026 EOL)	php r 'echo PHP_VERSION,' && systemctl restart php8.2 fpm
PHP 7.4, 8.0, 8.1	End of life (Apply vendor LTS package or minimal source backport immediately, then migrate)	Verify patched build timestamp and isolate behind WAF
PHP 7.3 and older	End of life with no upstream backport (Isolate, restrict access or migrate urgently)	Remove from direct public internet exposure

Table 18.1: Target patch versions and verification actions for October 2026

Always restart the worker processes

Upgrading the PHP package on disk does not update the code already loaded in RAM. After any package update, restart php fpm, Apache httpd, LiteSpeed or your application containers, and verify the loaded version from a web context rather than only from the command line.

18.3 Phase 3: Official interim mitigations when you cannot update today

The PHP security team made clear on 21 October 2026 and 23 October 2026 that only an update is a complete fix. However, if a legacy server cannot be updated for a few days while compatibility testing takes place, the PHP project published specific risk reduction steps for CVE 2026 109618, CVE 2026 109619 and the session upload progress vector. Apply these settings at the server or FPM pool level using `php_admin_value` or `php_admin_flag` so that application scripts cannot override them at runtime.

```

; === Interim risk reduction settings from the Oct 2026 PHP advisories ===
; Apply in php.ini or FPM pool config (e.g. /etc/php/8.x/fpm/pool.d/www.conf)
; Remember to restart all PHP FPM or Apache workers after saving.

; 1. Mitigation for CVE-2026-109619 and CVE-2026-115560 session paths:
; Disable session upload progress tracking completely.
session.upload_progress.enabled = 0

; 2. Mitigation for CVE-2026-109619 on PHP 8.5:
; Disable fatal error backtraces on unpatched 8.5 builds.
fatal_error_backtraces = 0

; 3. Mitigation for CVE-2026-109618 (OOM corruption path):
; Remove controllable PHP memory_limit bailout and bound request size at the proxy.
; Only set memory_limit = -1 if you enforce a container/cgroup limit that kills the
affected worker
; and you strictly limit request sizes at Nginx/Apache/WAF.
```

```
memory_limit = -1
post_max_size = 8M
upload_max_filesize = 4M
max_input_vars = 1000
max_input_nesting_level = 32
max_file_uploads = 5
```

If your application requires file uploads, the PHP advisory recommends setting `post_max_size` to 32M and keeping `upload_max_filesize` below `post_max_size`, while enforcing strict limits at the reverse proxy or web server on decoded request size, input field count, multipart part count, header count and nesting depth, including chunked requests that arrive without a Content Length header.

18.4 Phase 4: Web server and reverse proxy boundary controls

Place Nginx, Apache or a Web Application Firewall in front of PHP so oversized, malformed or exploit probing requests are rejected before the PHP interpreter ever sees them.

```
# Nginx reverse proxy protective boundary in /etc/nginx/conf.d/php-hardening.conf
client_max_body_size 8m;
client_body_buffer_size 128k;
client_header_buffer_size 1k;
large_client_header_buffers 4 8k;
client_body_timeout 10s;
client_header_timeout 10s;

# Strip PHP version disclosure headers at the proxy layer
fastcgi_hide_header X-Powered-By;
proxy_hide_header X-Powered-By;
server_tokens off;

# Block direct PHP execution inside upload and cache directories
location ~* ^/(wp-content/uploads|uploads|storage|cache|tmp)/.*\.(php|phtml|phar|inc)$ {
    deny all;
    return 403;
}

# Block public access to dotfiles, Composer metadata and backups
location ~* (\.(env|git|svn|htaccess|bak|swp|sql)|composer\.(json|lock)|phpinfo\.php)$ {
    deny all;
    return 404;
}
```

Chapter 19. Comprehensive PHP Hardening Reference for Production

Once emergency patching is complete, every production PHP server should be hardened so that even if an application bug is discovered in the future, the attacker cannot easily read configuration files, execute shell commands, move laterally or persist on the machine.

19.1 Production php.ini security baseline

The configuration block below represents Bithost's recommended baseline for production php.ini files on PHP 8.3, 8.4 and 8.5.

```
; === Bithost Production PHP Security Baseline (php.ini) ===

; Hide PHP version from HTTP headers
expose_php = Off

; Never display errors or stack traces to site visitors
display_errors = Off
display_startup_errors = Off
log_errors = On
error_log = /var/log/php/php_errors.log

; Disable remote file inclusion over HTTP/FTP wrappers
allow_url_fopen = Off
allow_url_include = Off

; Restrict dangerous OS command execution and inspection functions
disable_functions =
exec,passthru,shell_exec,system,proc_open,popen,pcntl_exec,dl,show_source,highlight_file,ph
pinfo

; Confine PHP filesystem access to the application directory and temp folder
open_basedir = /var/www/app:/tmp

; Harden session handling against hijacking and fixation
session.use_strict_mode = 1
session.use_only_cookies = 1
session.cookie_httponly = 1
session.cookie_secure = 1
session.cookie_samesite = "Lax"
session.name = "__Host-APPSESSID"
session.upload_progress.enabled = 0

; Bound input parsing to prevent memory and CPU exhaustion
max_execution_time = 30
max_input_time = 30
post_max_size = 16M
upload_max_filesize = 10M
max_file_uploads = 5
max_input_vars = 1000
```

```
max_input_nesting_level = 32

; Disable dynamic extension loading at runtime
enable_dl = Off

; Ensure phar archives remain read only
phar.readonly = On
```

19.2 PHP FPM pool isolation and systemd sandboxing

On a server running more than one PHP site, never run all sites inside a single shared www pool under the same Linux user. If Site A is compromised, it can read and overwrite every file belonging to Site B. Create a dedicated Linux user and a dedicated PHP FPM pool for each application, and use systemd unit directives to make the application code read only at runtime.

```
# /etc/systemd/system/php8.4-fpm.service.d/hardening.conf
[Service]
NoNewPrivileges=true
PrivateTmp=true
ProtectSystem=strict
ProtectHome=true
ProtectKernelTunables=true
ProtectKernelModules=true
ProtectControlGroups=true
RestrictSUIDSGID=true
ReadWritePaths=/var/www/app/storage /var/log/php /run/php
```

19.3 Container and Kubernetes security for PHP workloads

For teams running PHP inside Docker or Kubernetes, pin specific digest hashes or explicit patch tags such as php:8.4.27 fpm alpine rather than floating tags, run the container as a non root UID, mount the root filesystem read only, drop all Linux capabilities, and set an explicit container memory limit with a restart policy so an out of memory event terminates the single pod cleanly.

Chapter 20. Engineering Guide: How to Upgrade Legacy PHP Safely

Why do organisations put off upgrading from PHP 5.6 or PHP 7.4 to PHP 8.4 or 8.5? Because they fear breaking production. When done without a method, flipping the PHP version switch in production is indeed reckless. When done using the structured five step engineering process below, even a ten year old application can be upgraded predictably with zero unscheduled downtime.

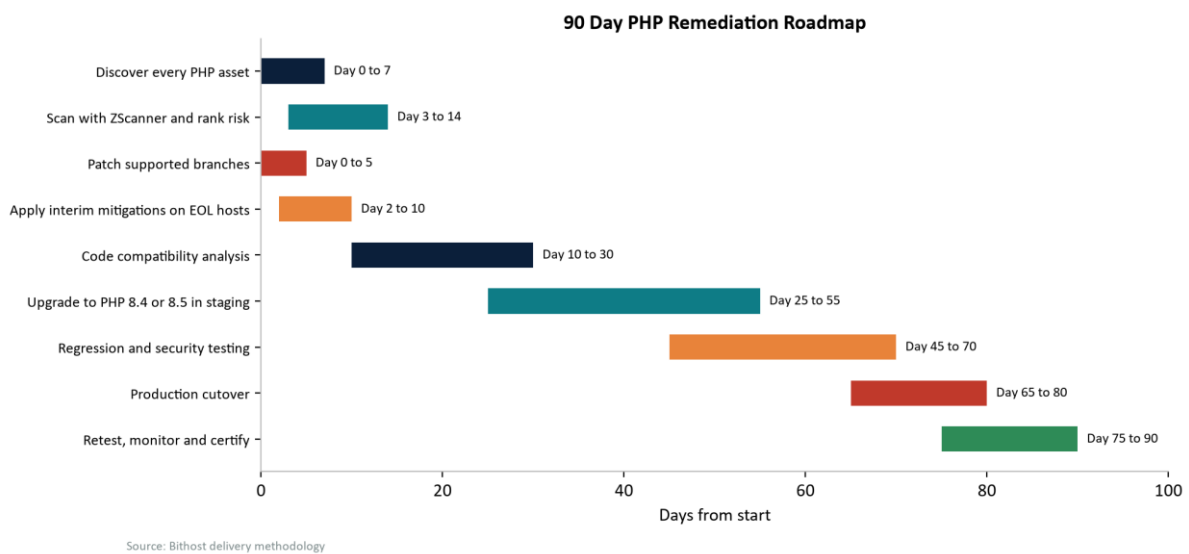


Figure 20.1: Bithost 90 day PHP remediation and upgrade roadmap

20.1 Major breaking changes between PHP 7.x and PHP 8.4 / 8.5

Transition	What changed in the language	How legacy code breaks and how to fix it
PHP 5.6 to 7.x	Removal of ext/mysql (mysql_query), ereg regex functions, and ext/mcrypt	Calls to mysql_connect() throw fatal errors. Replace with PDO or mysqli using prepared statements, and replace mcrypt with OpenSSL or Sodium.
PHP 7.4 to 8.0	Strict TypeError on internal functions, union types, string to number comparison changes, removal of create_function() and each()	Passing null or wrong types to strlen(), count() or array_merge() now throws a fatal TypeError instead of a warning. Validate inputs before calling built in functions.
PHP 8.0 to 8.1	Readonly properties, enums, Fibers, and deprecation of passing null to non nullable internal function parameters	Legacy code calling htmlspecialchars(\$val) when \$val is null emits deprecation floods. Use

Transition	What changed in the language	How legacy code breaks and how to fix it
		null coalescing htmlspecialchars(\$val ?? "").
PHP 8.1 to 8.2	Creation of dynamic class properties deprecated (#[AllowDynamicProperties] attribute needed), utf8_encode/decode deprecated	Classes that assign \$this->foo without declaring public \$foo in the class definition trigger deprecation notices. Declare properties explicitly.
PHP 8.2 to 8.3	Typed class constants, json_validate(), stricter Date/Time exceptions, unserialize() error handling adjustments	Check custom date parsers and subclasses that override parent class constants.
PHP 8.3 to 8.4 / 8.5	Property hooks, asymmetric visibility, implicit nullable parameter types deprecated (function foo(Bar \$a = null) must be ?Bar \$a = null)	Run automated syntax transforms to add explicit nullable question marks across method signatures.

Table 20.1: Key language changes from PHP 5.6 through PHP 8.5

20.2 Automated compatibility auditing and refactoring

Never inspect a hundred thousand lines of PHP by eye alone. Use static analysis and automated refactoring tools in your staging environment before running a single manual test.

```
# 1. Run PHP_CodeSniffer with the PHPCompatibility ruleset against your codebase
composer require --dev phpcompatibility/php-compatibility squizlabs/php_codesniffer
./vendor/bin/phpcs -p ./src ./app --standard=PHPCompatibility --runtime-set testVersion 8.4-

# 2. Run PHPStan to catch strict TypeErrors introduced in PHP 8
composer require --dev phpstan/phpstan
./vendor/bin/phpstan analyse ./src --level=5

# 3. Preview automated code upgrades using Rector
composer require --dev rector/rector
./vendor/bin/rector process ./src --dry-run
```

20.3 Blue green cutover with instant rollback

Instead of upgrading PHP in place on your live server, provision a parallel server or PHP FPM socket running PHP 8.4.27 or PHP 8.5.12 alongside the existing PHP pool. Replay a sample of production read requests against the new pool, inspect the error logs, and then switch the Nginx upstream block or load balancer target. If an unexpected edge case appears on a specific legacy URL, you can route that single route or the entire site back in five seconds while fixing the line of code.

Chapter 21. Complete php.ini Security Directive Reference

During security audits, administrators often ask which exact settings in php.ini control security behavior, what PHP sets them to by default, and what value should be used in production. Table 29.1 provides a complete forty directive reference covering information disclosure, memory safety, request bounds, filesystem confinement, session management, cryptography and OPcache hardening.

Directive name	Default value	Recommended production value	Security purpose and operational notes
expose_php	On	Off	Stops PHP from sending the X Powered By HTTP header with the exact runtime version on every response.
display_errors	On (dev) / Off	Off	Prevents stack traces, file paths, database hostnames and variable contents from printing to the browser.
display_startup_errors	On (dev) / Off	Off	Hides module load failures and internal initialization warnings from remote visitors.
log_errors	Off / On	On	Ensures all runtime warnings,

Directive name	Default value	Recommended production value	Security purpose and operational notes
			notices and fatal errors are written to a protected server log file.
error_log	Empty	/var/log/php/error.log	Routes PHP errors to a dedicated directory outside the public web document root.
ignore_repeated_errors	Off	On	Prevents a single looped warning from filling the disk partition during an active probing flood.
allow_url_fopen	On	Off (or On only if required)	Blocks file_get_contents(), include() and copy() from fetching remote URLs over HTTP or FTP wrappers.
allow_url_include	Off	Off	Must never be enabled. Prevents remote file inclusion attacks where

Directive name	Default value	Recommended production value	Security purpose and operational notes
			remote text is executed as PHP code.
enable_dl	Off	Off	Prevents scripts from loading arbitrary compiled C shared libraries at runtime using dl().
open_basedir	Empty	/var/www/app:/tmp	Confines PHP file operations to the specified directory trees so a compromised script cannot read /etc or neighbor sites.
disable_functions	Empty	exec,passthru,shell_exec,system,proc_open,popen,pcntl_exec,dls,how_source,highlight_file,phpinfo	Blocks direct operating system command execution and configuration dumping functions inside web workers.
max_execution_time	30	30	Bounds CPU seconds a single script may run before termination, limiting

Directive name	Default value	Recommended production value	Security purpose and operational notes
			denial of service impact.
max_input_time	60	30	Bounds the time PHP spends parsing POST, GET and multipart file upload payloads.
memory_limit	128M	128M to 256M (or 1 with strict cgroup & proxy bounds per CVE 2026 109618 advisory)	Controls per worker memory allocation. Pair with container/cgroup limits and strict proxy request size bounds.
post_max_size	8M	8M (no uploads) or 32M (with uploads)	Caps total size of an HTTP POST body. Key boundary control highlighted in the Oct 2026 PHP advisory.
upload_max_filesize	2M	4M to 16M (always below post_max_size)	Caps the maximum size of any single uploaded file.
max_file_uploads	20	5	Limits how many files can be

Directive name	Default value	Recommended production value	Security purpose and operational notes
			uploaded in a single multipart request.
max_input_vars	1000	1000	Limits the number of GET, POST and COOKIE input variables parsed per request to prevent hash collision floods.
max_input_nesting_level	64	32	Limits how deeply nested input arrays can be, reducing parser stack and memory pressure.
file_uploads	On	Off (if app never accepts files) or On	Turn off completely on APIs or brochure sites that never require user file uploads.
upload_tmp_dir	System default	/var/www/app/tmp_uploads	Places temporary multipart uploads in a dedicated non executable directory with strict quota limits.

Directive name	Default value	Recommended production value	Security purpose and operational notes
session.upload_progress.enabled	On (1)	Off (0)	Critical mitigation from the 21 and 23 Oct 2026 PHP advisories for CVE 2026 109619 and CVE 2026 115560.
fatal_error_backtraces	On (PHP 8.5)	Off (0) on unpatched 8.5	Blocks the confirmed fatal error backtrace path for CVE 2026 109619 on PHP 8.5 builds.
session.save_handler	files	redis or files (in isolated 0700 path)	Stores session data in an authenticated Redis instance or a strictly private directory.
session.save_path	/tmp	/var/lib/php/sessions/app_pool	Prevents other Linux users on the same host from listing or reading session files in shared /tmp.
session.use_strict_mode	0	1	Forces PHP to reject uninitialized session IDs supplied by an attacker, preventing

Directive name	Default value	Recommended production value	Security purpose and operational notes
			session fixation.
<code>session.use_cookies</code>	1	1	Uses HTTP cookies rather than URL query parameters to transport session identifiers.
<code>session.use_only_cookies</code>	1	1	Forbids PHP from accepting session IDs passed in URL query strings.
<code>session.use_trans_sid</code>	0	0	Prevents PHP from automatically rewriting links to include session IDs in URLs.
<code>session.cookie_httponly</code>	0	1	Marks the session cookie HttpOnly so injected JavaScript cannot steal it via document.cookie.
<code>session.cookie_secure</code>	0	1	Ensures the browser transmits the session cookie only over encrypted

Directive name	Default value	Recommended production value	Security purpose and operational notes
			HTTPS connections.
session.cookie_samesite	Empty	Lax or Strict	Protects authenticated sessions against cross site request forgery (CSRF) across domains.
session.name	PHPSESSID	__Host APPSESSID	Replaces the fingerprinting default name and enforces HTTPS and root path binding via the __Host prefix.
session.sid_length	32	48	Increases session identifier entropy on PHP branches below 8.4 (standard 32 byte CSPRNG in 8.4+).
session.gc_maxlifetime	1440	900 to 1800	Expires idle server side session data after 15 to 30 minutes of inactivity.
phar.readonly	On	On	Prevents PHP scripts from creating or

Directive name	Default value	Recommended production value	Security purpose and operational notes
			modifying .phar archives in production.
zend.exception_ignore_args	Off / On	On	Excludes function argument values (such as passwords or card numbers) from exception stack traces.
zend.assertions	1	1	Compiles out assert() calls completely in production with zero runtime overhead.
opcache.enable	1	1	Caches precompiled bytecode in shared memory for speed and stability.
opcache.validate_timestamps	1	0 (in immutable containers)	When set to 0 in read only container deployments, PHP never re-reads .php files from disk even if an attacker writes a file.

Table 21.1: Forty essential php.ini directives for production security and stability

Chapter 22. Server Architecture Hardening Blueprints

As highlighted in the 21 October 2026 PHP advisory, PHP runs across several different server architectures, including Apache with mod_php, PHP FPM behind Nginx or Apache, php cgi, FrankenPHP and LiteSpeed. Each architecture requires specific configuration changes to eliminate high risk paths and contain potential compromises.

22.1 Migrating away from Apache mod_php to isolated PHP FPM

In the October 2026 PHP2Shell advisory, the PHP team explicitly noted that risk is highest for Apache with mod_php, where a demonstrated exploit allows an unauthenticated request to trigger the attack before any application code runs. Beyond that specific bug, mod_php embeds the PHP interpreter inside every Apache worker process, meaning static image and CSS requests consume heavy PHP memory and all sites on the server share the same Unix user.

Every organisation still running mod_php should switch Apache to the event MPM paired with PHP FPM over a Unix domain socket. On Ubuntu or Debian servers, the transition takes four commands after installing the patched PHP 8.4 FPM package.

```
# Switch Apache from prefork + mod_php to event MPM + php8.4-fpm
sudo apt-get update && sudo apt-get install -y php8.4-fpm libapache2-mod-fcgid
sudo a2dismod php7.4 php8.0 php8.1 php8.2 mpm_prefork
sudo a2enmod mpm_event proxy proxy_fcgi setenvif headers
sudo a2enconf php8.4-fpm
sudo systemctl restart php8.4-fpm apache2
```

22.2 Hardening PHP FPM pool configurations

When configuring a PHP FPM pool in /etc/php/8.4/fpm/pool.d/app.conf, use php_admin_value and php_admin_flag rather than regular php_value directives. Settings applied with php_admin_value cannot be overridden by ini_set() calls in application code or by malicious .user.ini files uploaded by an attacker.

```
[app_production]
user = app_worker
group = app_worker
listen = /run/php/php8.4-fpm-app.sock
listen.owner = www-data
listen.group = www-data
listen.mode = 0660

; Process manager sizing and periodic worker recycling
pm = dynamic
```

```
pm.max_children = 40
pm.start_servers = 8
pm.min_spare_servers = 4
pm.max_spare_servers = 16
pm.max_requests = 1000
request_terminate_timeout = 30s

; Clean environment variables so host secrets do not leak into worker memory
clear_env = yes

; Non overridable security settings (including Oct 2026 PHP2Shell mitigations)
php_admin_flag[expose_php] = off
php_admin_flag[display_errors] = off
php_admin_flag[log_errors] = on
php_admin_value[error_log] = /var/log/php/app-error.log
php_admin_value[session.upload_progress.enabled] = 0
php_admin_value[open_basedir] = /var/www/app:/tmp
php_admin_value[disable_functions] =
exec, passthru, shell_exec, system, proc_open, popen, pcntl_exec, dl, phpinfo
php_admin_value[post_max_size] = 8M
php_admin_value[upload_max_filesize] = 4M
```

22.3 Hardening Docker and Kubernetes PHP deployments

Containerised PHP workloads often suffer from a different flaw: developers build a Docker image once using FROM php:8.1 fpm, push it to a container registry, and run that exact unpatched image in Kubernetes for two years without rebuilding it. Below is a hardened multi stage Dockerfile and Kubernetes securityContext pattern for PHP 8.4.27+.

```
# Hardened production Dockerfile pinned to patched PHP 8.4.27
FROM php:8.4.27-fpm-alpine AS production

# Install security updates in base Alpine packages
RUN apk update && apk upgrade --no-cache

# Copy hardened php.ini and fpm pool config
COPY docker/php/security.ini /usr/local/etc/php/conf.d/99-security.ini
COPY docker/php/www.conf /usr/local/etc/php-fpm.d/zz-hardened.conf

# Copy application code owned by root, readable by www-data (read only code tree)
WORKDIR /var/www/app
COPY --chown=root:www-data . /var/www/app
RUN chmod -R 0750 /var/www/app && chown -R www-data:www-data /var/www/app/storage

USER www-data
EXPOSE 9000
CMD ["php-fpm", "-F"]
```

In Kubernetes or Docker Compose, pair the image above with a container memory limit and restart policy so that if an out of memory condition occurs, the kernel cgroup kills only the affected worker or pod cleanly.

22.4 Hardening Windows IIS and FastCGI PHP servers

For organisations running PHP on Windows Server with IIS, never map *.php requests to php cgi.exe in plain CGI mode. Configure IIS to use the FastCGI module exclusively, upgrade the Windows PHP binaries from windows.php.net to PHP 8.5.12, 8.4.27, 8.3.36 or 8.2.35, block HTTP query strings containing raw 0xAD soft hyphen or encoded %AD sequences in IIS Request Filtering, and audit the host with Bithost EWSAF.

Chapter 23. Web Application Firewall and Web Server

Rulebook for PHP

While patching the PHP runtime is always the primary fix, a defense in depth architecture stops exploit payloads at the network edge before they reach PHP worker processes. The rules in this chapter can be deployed on ModSecurity, OWASP Coraza, Nginx, Apache and Windows IIS.

23.1 ModSecurity and Coraza WAF rules for PHP2Shell and CGI attacks

Add the rules below to your ModSecurity or Coraza configuration to block the specific HTTP patterns used in CVE 2026 109618, CVE 2026 109619, CVE 2026 115560 and CVE 2024 4577.

```
# Rule 990001: Block PHP_SESSION_UPLOAD_PROGRESS abuse (CVE-2026-109619 / CVE-2026-115560)
SecRule ARGS_NAMES|FILES_NAMES|REQUEST_BODY "@rx (?i)PHP_SESSION_UPLOAD_PROGRESS" \
    "id:990001,phase:2,deny,status:403,log,msg:'Blocked PHP Session Upload Progress exploit attempt'"

# Rule 990002: Block Windows PHP-CGI Soft Hyphen 0xAD argument injection (CVE-2024-4577)
SecRule QUERY_STRING|REQUEST_URI_RAW "@rx
(?i)(?:%ad|\xad)+(?:d|allow_url_include|auto_prepend_file)" \
    "id:990002,phase:1,t:none,deny,status:403,log,msg:'Blocked CVE-2024-4577 PHP-CGI argument injection'"

# Rule 990003: Block php://input, phar://, expect:// and data:// stream wrapper injection
SecRule ARGS|ARGS_NAMES|REQUEST_COOKIES|REQUEST_HEADERS "@rx
(?i)(?:php://(?:input|filter)|phar://|expect://|data://text)" \
    "id:990003,phase:2,t:urlDecodeUni,deny,status:403,log,msg:'Blocked dangerous PHP stream wrapper in request'"

# Rule 990004: Block external probes for forgotten phpinfo, .env and Composer files
SecRule REQUEST_FILENAME "@rx
(?i)(?:/(?:phpinfo|info|test|adminer)\.php|/\.(?:env|git/config)|/composer\.(?:json|lock))$" \
    "id:990004,phase:1,t:lowercase,deny,status:404,log,msg:'Blocked probe for sensitive PHP diagnostic or config file'"

```

23.2 Apache .htaccess and VirtualHost drop in protection

For sites running on Apache where administrators configure security inside VirtualHost blocks or .htaccess files, apply the template below to hide PHP banners, block upload directory script execution and reject suspicious query strings.

```
# Apache VirtualHost / .htaccess Hardening Template for PHP Estates
ServerSignature Off
Header always unset X-Powered-By
Header unset X-Powered-By

```

```
# Security response headers
Header always set X-Content-Type-Options "nosniff"
Header always set X-Frame-Options "SAMEORIGIN"
Header always set Referrer-Policy "strict-origin-when-cross-origin"

# Block CVE-2024-4577 soft hyphen and dangerous PHP wrapper strings
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{QUERY_STRING}
    (?i)(%ad|\xad|allow_url_include|auto_prepend_file|php://input|phar://) [NC]
    RewriteRule .* - [F,L]
</IfModule>

# Block access to .env, .git, Composer files and backup extensions
<FilesMatch
" (?i)(^\. (env|git|htaccess)|composer\. (json|lock)|\. (bak|swp|old|sql|tar|gz|zip)|^(phpinfo|
info)\.php)$">
    Require all denied
</FilesMatch>
```

Chapter 24. Framework by Framework Upgrade Playbooks

Upgrading a raw PHP script is different from upgrading a large WordPress store, a Laravel SaaS application, a Drupal government portal or a Magento commerce platform. This chapter provides concrete upgrade checklists for the six most common PHP application families.

24.1 WordPress and WooCommerce upgrade playbook

1. Inventory every active and inactive plugin and theme using WP CLI (`wp plugin list` and `wp theme list`). Delete every inactive plugin and theme immediately.
2. Clone the production files and database to a staging environment running alongside PHP 8.4.27 or PHP 8.5.12.
3. Enable `WP_DEBUG` and `WP_DEBUG_LOG` in staging `wp config.php` while keeping `WP_DEBUG_DISPLAY` set to false.
4. Update WordPress core, WooCommerce and all actively maintained plugins to their latest releases in staging.
5. Scan the `wp content/plugins` and `wp content/themes` directories using PHPCompatibility and Bithost CodeSecurity to flag plugins calling removed PHP 7 functions or creating dynamic class properties.
6. Replace any abandoned plugin that has not been updated in the last twenty four months with an actively maintained equivalent, or refactor custom functions in your child theme.
7. Test critical user journeys in staging: customer registration, product search, add to cart, coupon calculation, payment gateway webhook callback, PDF invoice generation and admin order export.
8. Switch the production PHP FPM pool to PHP 8.4.27 or PHP 8.5.12, flush OPcache and object cache, and run a full ZScanner verification scan.

24.2 Laravel and Symfony upgrade playbook

1. Inspect `composer.json` and run `composer outdated` and `composer audit` to see which dependencies block PHP 8.4 or 8.5.
2. Use Rector with the Laravel or Symfony rulesets alongside PHPStan to automate syntax and type signature updates across `app/` and `src/` directories.
3. Pay special attention to null safety: PHP 8.1+ deprecates passing null to internal string and math functions, and PHP 8.4 deprecates implicit nullable parameter declarations.
4. Replace abandoned third party packages (for example moving from `swiftmailer/swiftmailer` to `symfony/mailer`, or from `phpoffice/phpexcel` to `phpoffice/phpspreadsheet`).

5. Update your Dockerfile or provisioning scripts to install PHP 8.4.27 or 8.5.12 along with required extensions (intl, mbstring, pdo_mysql, redis, opcache, zip).
6. Run your automated PHPUnit or Pest test suite under the new runtime and inspect deprecation logs before blue green deployment.

24.3 Drupal 7 / 9 / 10 to Drupal 10 / 11 on PHP 8.4 playbook

1. For legacy Drupal 7 sites, note that Drupal 7 has reached end of community support. Isolate the site behind strict WAF rules immediately while executing a structured content and module migration to Drupal 10 or 11 on PHP 8.3/8.4.
2. For Drupal 9 and 10 sites, run the Upgrade Status module to identify contributed and custom modules with deprecated API calls or PHP 8.4 incompatibilities.
3. Update Composer constraints, apply drupal rector patches to custom modules, test Twig templates for strict variable handling, and deploy on patched PHP 8.4.27 or 8.3.36.

24.4 Magento and OpenMage upgrade playbook

1. Audit all third party payment, shipping, tax and ERP modules in app/code and vendor/ against the target PHP 8.2.35, 8.3.36 or 8.4.27 compatibility matrix.
2. For legacy Magento 1 stores that cannot afford a full replatforming project this quarter, migrate onto actively maintained OpenMage LTS builds that support PHP 8.2+ rather than leaving the server on unpatched PHP 7.2 or 7.4.
3. Test checkout webhooks, cron indexers, Elasticsearch/OpenSearch catalog indexing and message queues in staging before switching production traffic.

24.5 Moodle learning management system upgrade playbook

1. Check the official Moodle environment compatibility matrix (Site administration > Server > Environment) to verify the minimum and maximum supported PHP version for your target Moodle LTS release.
2. Audit all custom question types, plagiarism plugins, video conferencing connectors and authentication modules in a staging clone before semester break.
3. Upgrade the PHP runtime to a patched release (8.3.36 or 8.4.27), run `php admin/cli/upgrade.php`, purge caches and verify cron execution.

24.6 Custom legacy PHP (CodeIgniter, CakePHP or raw procedural PHP) playbook

1. Place the codebase under Git version control if it is not already tracked, and scan the entire tree with Bithost CodeSecurity to find hardcoded credentials, SQL injection and removed functions.

2. Write a lightweight smoke test script using curl or Playwright that exercises the top thirty URLs and form submissions of the application and records HTTP status codes and response lengths.
3. Fix the three biggest blockers when moving from PHP 5/7 to PHP 8.4: replace `mysql_*` calls with PDO prepared statements, replace `mcrypt_*` calls with `openssl_encrypt/decrypt` or `sodium_crypto_secretbox`, and initialize variables and array keys before reading them so undefined array key warnings and `TypeError`s do not break pages.
4. Deploy behind a hardened Nginx + PHP FPM 8.4.27 stack with `session.upload_progress.enabled=0` and `expose_php=Off`.

Chapter 25. Complete Code Migration Reference: PHP 5.6 and 7.x to PHP 8.4 and 8.5

When engineering teams estimate how long a PHP upgrade will take, the biggest unknown is always code remediation. How many files in the repository use functions or language behaviours that changed between PHP 5.6, PHP 7.4 and PHP 8.5? This chapter documents every common code pattern that breaks during an upgrade and shows the exact before and after code that fixes it.

25.1 Function and extension replacement catalog

Legacy construct or extension	Removed or deprecated in	Why it was removed	Modern replacement in PHP 8.4 / 8.5
<code>mysql_connect()</code> , <code>mysql_query()</code> , <code>mysql_real_escape_string()</code>	Removed in PHP 7.0	Insecure procedural API lacking native prepared statements and modern character set handling	PDO (new <code>PDO()</code>) or <code>mysqli</code> with prepared statements and bound parameters
<code>ereg()</code> , <code>eregi()</code> , <code>ereg_replace()</code> , <code>split()</code>	Removed in PHP 7.0	Slow POSIX regular expression library superseded by PCRE	<code>preg_match()</code> , <code>preg_replace()</code> , <code>preg_split()</code> with PCRE delimiters
<code>preg_replace()</code> with <code>/e</code> eval modifier	Removed in PHP 7.0	Severe remote code execution risk whenever user input reached the replacement pattern	<code>preg_replace_callback()</code> using an anonymous closure
<code>mdecrypt_encrypt()</code> , <code>mdecrypt_decrypt()</code> , <code>mdecrypt_create_iv()</code>	Removed in PHP 7.2	Unmaintained C library with insecure default block modes and padding	<code>sodium_crypto_secretbox()</code> (libsodium) or <code>openssl_encrypt()</code> with AES 256 GCM
<code>create_function()</code>	Deprecated in 7.2, removed in 8.0	Internally used <code>eval()</code> on strings, creating code injection vectors and memory leaks	Native anonymous functions: <code>function(\$a) use (\$b) { return \$a + \$b; }</code> or <code>fn(\$a) => \$a + \$b</code>
<code>each()</code>	Deprecated in 7.2, removed in 8.0	Slow internal array cursor iteration rarely needed in modern code	Standard <code>foreach (\$array as \$key => \$value)</code> loop
<code>__autoload()</code> global function	Deprecated in 7.2, removed in 8.0	Only allowed a single autoloader across the entire application	<code>spl_autoload_register()</code> or Composer PSR 4 autoloader

Legacy construct or extension	Removed or deprecated in	Why it was removed	Modern replacement in PHP 8.4 / 8.5
parse_str() without second argument	Deprecated in 7.2, removed in 8.0	Extracted query string variables directly into the current symbol table, equivalent to register_globals	Always pass the target array: parse_str(\$queryString, \$outputArray)
assert() with string argument	Deprecated in 7.2, removed in 8.0	Evaluated strings as PHP code, allowing remote code execution	Pass a boolean expression directly: assert(\$value > 0)
get_magic_quotes_gpc(), get_magic_quotes_runtime()	Removed in PHP 8.0	Magic quotes were removed years earlier in PHP 5.4	Remove calls completely and rely on context specific escaping or prepared statements
implode(\$pieces, \$glue) reversed parameter order	Removed in PHP 8.0	Historical quirk that confused static analysers	Always pass glue string first: implode(\$glue, \$pieces)
money_format()	Removed in PHP 8.0	Relied on OS specific C strfmon library missing on Windows and inconsistent across locales	NumberFormatter::formatCurrency() from the intl extension
strftime() and gmstrftime()	Deprecated in PHP 8.1	Inconsistent locale behaviour across operating systems and thread safety issues	IntlDateFormatter::format() or DateTimeInterface::format()
utf8_encode() and utf8_decode()	Deprecated in PHP 8.2	Misleading names that only converted between ISO 8859 1 and UTF 8	mb_convert_encoding(\$str, 'UTF 8', 'ISO 8859 1') or UConverter::transcode()
Dynamic class properties (\$this > undeclared = 1)	Deprecated in PHP 8.2	Masked typos in property names and increased object memory overhead	Declare properties explicitly on the class, or use #[AllowDynamicProperties] during transition
Partially supported callables such as "self::method"	Deprecated in PHP 8.2	Inconsistent behaviour between call_user_func(\$c) and \$c()	Use [self::class, 'method'] or first class callable syntax self::method(...)
\${var} string interpolation syntax	Deprecated in PHP 8.2	Ambiguous dollar brace placement inside double quoted strings	Use standard {\$var} or \$var interpolation inside double quoted strings
Implicitly nullable parameter types (function f(Foo \$a = null))	Deprecated in PHP 8.4	Ambiguity in type system reflection when a typed	Mark nullable types explicitly with a leading question mark: function f(?Foo \$a = null)

Legacy construct or extension	Removed or deprecated in	Why it was removed	Modern replacement in PHP 8.4 / 8.5
		parameter defaults to null	
session_set_save_handler() with 6 procedural callbacks	Deprecated in PHP 8.4	Procedural callback registration is error prone compared to interface contracts	Pass an object implementing SessionHandlerInterface

Table 25.1: Comprehensive migration matrix of functions and language features from PHP 5.6 to PHP 8.5

25.2 Before and after refactoring recipes

Developers upgrading legacy applications can use the six side by side patterns below as templates during code review.

Recipe 1: Replacing legacy `mysql_*` calls with PDO prepared statements

Legacy PHP 5 code often concatenates request variables directly into SQL strings passed to `mysql_query()`. That pattern both fails completely on PHP 7 and 8 and exposes the database to SQL injection.

```
// LEGACY INSECURE CODE (Fails in PHP 7.0+ and vulnerable to SQL injection)
$conn = mysql_connect('localhost', 'dbuser', 'dbpass');
mysql_select_db('customers', $conn);
$id = $_GET['id'];
$res = mysql_query("SELECT id, name, email FROM users WHERE id = " . $id);
$row = mysql_fetch_assoc($res);

// MODERN SECURE CODE FOR PHP 8.4 / 8.5
$dsn = 'mysql:host=127.0.0.1;dbname=customers;charset=utf8mb4';
$pdo = new PDO($dsn, $dbUser, $dbPass, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false,
]);
$stmt = $pdo->prepare('SELECT id, name, email FROM users WHERE id = :id');
$stmt->execute(['id' => (int) ($_GET['id'] ?? 0)]);
$row = $stmt->fetch();
```

Recipe 2: Replacing removed `mcrypt` encryption with `libsodium` or `OpenSSL AES GCM`

Many older billing and healthcare applications used `ext/mcrypt` to encrypt tax numbers or API credentials in the database. In PHP 8.4 and 8.5, `libsodium` is built into core PHP and provides authenticated encryption out of the box.

```
// LEGACY CODE (Fails in PHP 7.2+ because ext/mcrypt was removed)
$iv = mcrypt_create_iv(mcrypt_get_iv_size(MCRYPT_RIJNDAEL_128, MCRYPT_MODE_CBC),
    MCRYPT_RAND);
$cipher = base64_encode($iv . mcrypt_encrypt(MCRYPT_RIJNDAEL_128, $key, $plaintext,
    MCRYPT_MODE_CBC, $iv));

// MODERN AUTHENTICATED ENCRYPTION IN PHP 8.4 / 8.5 (libsodium)
$nonce = random_bytes(SODIUM_CRYPT_SECRETBOX_NONCEBYTES);
$ciphertext = sodium_crypto_secretbox($plaintext, $nonce, $key32Bytes);
$storedPayload = base64_encode($nonce . $ciphertext);

// Decryption with automatic tamper verification
$decoded = base64_decode($storedPayload, true);
$nonce = substr($decoded, 0, SODIUM_CRYPT_SECRETBOX_NONCEBYTES);
$box = substr($decoded, SODIUM_CRYPT_SECRETBOX_NONCEBYTES);
$decrypted = sodium_crypto_secretbox_open($box, $nonce, $key32Bytes);
if ($decrypted === false) {
    throw new RuntimeException('Ciphertext verification failed');
}
```

Recipe 3: Fixing PHP 8.1+ null deprecation and TypeError on internal string functions

In PHP 7.4, passing null to `strlen()`, `trim()`, `htmlspecialchars()` or `explode()` silently cast null to an empty string. In PHP 8.1 and newer, passing null to non nullable internal parameters emits deprecation notices and leads to strict TypeErrors when `strict_types` is enabled.

```
// LEGACY CODE (Triggers deprecation flood in PHP 8.1 to 8.5)
$username = trim($_POST['username']);
$safeBio = htmlspecialchars($userRow['bio']);
$items = count($order['line_items']);

// MODERN NULL SAFE CODE FOR PHP 8.4 / 8.5
$username = trim((string) ($_POST['username'] ?? ''));
$safeBio = htmlspecialchars((string) ($userRow['bio'] ?? ''), ENT_QUOTES | ENT_SUBSTITUTE,
    'UTF-8');
$items = is_countable($order['line_items'] ?? null) ? count($order['line_items']) : 0;
```

Recipe 4: Fixing PHP 8.2 dynamic properties and PHP 8.4 implicit nullable types

Two syntax cleanups account for most warnings when moving from PHP 8.0 or 8.1 to PHP 8.4 and 8.5: declaring class properties explicitly and marking nullable typed parameters with a question mark.

```
// LEGACY CODE (Emits dynamic property deprecation in 8.2+ and implicit nullable
deprecation in 8.4+)
class InvoiceService
{
    public function __construct(LoggerInterface $logger = null)
    {
        $this->logger = $logger; // Undeclared property $logger
    }
}
```

```
}  
  
// MODERN CLEAN CODE FOR PHP 8.4 / 8.5 (Constructor property promotion + explicit  
?LoggerInterface)  
class InvoiceService  
{  
    public function __construct(  
        private ?LoggerInterface $logger = null,  
    ) {}  
}
```

Chapter 26. Automated CI/CD Security Gates for PHP Engineering Teams

Fixing outdated PHP once is good. Building an automated deployment pipeline that refuses to ship outdated runtimes, vulnerable Composer packages or insecure code patterns is much better. Below is a complete GitHub Actions or GitLab CI workflow pattern that combines PHP 8.4/8.5 compatibility checks, Composer dependency auditing, Bithost CodeSecurity scanning and post deployment ZScanner verification.

```
# .github/workflows/php-security-and-upgrade-gate.yml
name: PHP 8.4 & 8.5 Security Gate

on:
  push:
    branches: [ main, staging ]
  pull_request:
    branches: [ main ]
  schedule:
    - cron: '0 3 * * 1' # Weekly supply chain & runtime check every Monday at 03:00 UTC

jobs:
  verify-php-runtime-and-code:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        php-version: ['8.4', '8.5']
    steps:
      - name: Checkout source repository
        uses: actions/checkout@v4

      - name: Provision patched PHP runtime
        uses: shivammathur/setup-php@v2
        with:
          php-version: ${matrix.php-version}
          extensions: mbstring, intl, pdo_mysql, sodium, opcache, zip
          ini-values: expose_php=Off, session.upload_progress.enabled=0

      - name: Verify minimum patched patch level for PHP2Shell
        run: |
          php -r '
            $v = PHP_VERSION;
            echo "Running PHP $v\n";
            if (version_compare($v, "8.4.27", "<")) {
              fwrite(STDERR, "ERROR: PHP runtime below Oct 2026 security release!\n");
              exit(1);
            }
          '

      - name: Install Composer dependencies with lockfile verification
        run: composer install --no-interaction --prefer-dist --no-progress
```

```
8.4-  
- name: Audit Composer packages for known CVEs  
  run: composer audit --format=plain  
  
- name: Run PHPCompatibility & PHPStan static type analysis  
  run: |  
    ./vendor/bin/phpcs -p ./src --standard=PHPCompatibility --runtime-set testVersion  
    ./vendor/bin/phpstan analyse ./src --level=6 --no-progress  
  
- name: Execute automated unit and integration test suite  
  run: ./vendor/bin/phpunit --fail-on-deprecation --fail-on-warning
```

Pair the repository workflow above with Bithost CodeSecurity (bithost.in/codesecurity) on your build server to inspect every commit for SAST vulnerabilities, hardcoded secrets, Dockerfile misconfigurations and CycloneDX SBOM updates.

Chapter 27. Post Compromise Forensics and Threat Hunting for PHP Servers

If your server was running an unpatched PHP version on the public internet after 21 October 2026, or if it ran PHP 7.x or unpatched Windows CGI during earlier exploitation waves, patching the binary is only half the job. You must also verify that no attacker established persistence before the patch went live.

27.1 Hunting for web shells and modified PHP files

Run the diagnostic commands below on every production PHP server to spot newly created scripts, hidden backdoors in upload directories and suspicious function calls.

```
# 1. Find any .php, .phtml or .phar files modified in the last 14 days
find /var/www -type f \( -name '*.php' -o -name '*.phtml' -o -name '*.phar' -o -name
'.user.ini' -o -name '.htaccess' \) -mtime -14 -ls

# 2. Find any PHP files sitting inside upload, media, cache or temp directories
find /var/www -regextype posix-extended -iregex
'.*(uploads?|media|images|cache|tmp|storage)/.*\.(php[0-9]?|phtml|phar|inc)$' -ls

# 3. Search for common obfuscated web shell patterns (eval + base64_decode, gzinflate,
str_rot13, assert)
grep -rnE
'(eval|assert|passthru|shell_exec|system|proc_open)\s*\(\s*(base64_decode|gzinflate|str_rot
13|gzuncompress|\$_POST|\$_GET|\$_REQUEST)' /var/www/

# 4. Check for unauthorized auto_prepend_file or auto_append_file hooks
grep -rnE 'auto_(prepend|append)_file' /etc/php* /var/www/
```

27.2 Inspecting access and error logs for PHP2Shell and CGI exploitation

Exploitation attempts against CVE 2026 109618, CVE 2026 109619 and CVE 2024 4577 leave characteristic signatures in Nginx, Apache, PHP FPM and system logs.

Log source	Indicator to look for	What it signals
System kernel / dmesg / journalctl	Out of memory: Killed process or php fpm worker exited on signal 11 (SIGSEGV) or signal 6 (SIGABRT)	Repeated worker crashes or OOM events triggered by CVE 2026 109618 memory corruption probing
Nginx / Apache access.log	POST requests containing PHP_SESSION_UPLOAD_PROGRESS multipart fields paired with	Active probing of CVE 2026 109619 and CVE 2026 115560 session upload progress paths

Log source	Indicator to look for	What it signals
	anomalous request sizes or chunked transfers	
Nginx / Apache access.log	Query strings containing %ADd, %add+allow_url_include%3d1 or auto_prepend_file%3dphp%3a//input	Exploitation attempts targeting CVE 2024 4577 PHP CGI argument injection
PHP error.log	Allowed memory size of X bytes exhausted repeated rapidly from the same source IP or subnet	Automated attacker tuning payload sizes to hit the exact OOM bailout boundary
Linux auditd / auth.log	Child processes (/bin/sh, curl, wget, python, nc, whoami, id) spawned with parent process php fpm, apache2 or httpd	Confirmed remote code execution inside a PHP worker process

Table 27.1: Forensic indicators of PHP runtime exploitation across server logs

27.3 Secret rotation and clean recovery procedure

Because CVE 2026 109619 and related memory flaws can expose sensitive worker memory across requests without leaving a file on disk, any server that shows signs of active exploitation should undergo five recovery steps.

1. Preserve forensic evidence by taking an immediate disk snapshot and archiving access, error and system logs to immutable storage.
2. Rebuild the virtual machine or container from a clean base image with PHP 8.5.12, 8.4.27, 8.3.36 or 8.2.35 rather than trying to clean a compromised server in place.
3. Rotate every secret that was accessible to the PHP process: database passwords, APP_KEY / JWT signing keys, cloud IAM access keys, payment gateway keys and SMTP credentials.
4. Invalidate all active user and administrator sessions in Redis, Memcached, database or file session stores so stolen session cookies cannot be replayed.
5. If personal data or unauthorized system access occurred, evaluate statutory reporting obligations (including the 6 hour CERT In timeline and DPDPA 2023 breach obligations in India, or 72 hour GDPR notification in Europe).

Chapter 28. Executive Briefing Pack: Questions Every Board Should Ask This Week

Chief executives, board audit committees and non technical directors do not need to read php.ini files, yet they carry ultimate accountability for the financial, operational and legal consequences of a breach. We close the main text of this report with ten clear questions that every board or leadership team can send to their technology head today, along with what a healthy answer looks like.

No.	Question for the CIO, CTO or IT Head	Red flag answer	Healthy, mature answer
1	How many internet facing websites, portals and APIs do we operate that use PHP, including subsidiaries and agency built microsites?	We think we have two or three main sites, we do not track what marketing agencies host for us.	We have a complete inventory of 28 domains across main sites, portals and microsites, verified by external scanning.
2	How many of those servers are currently running end of life PHP (PHP 8.1, 8.0, 7.x or 5.x) or PHP 8.2 which expires on 31 December 2026?	We are not sure of the exact version numbers on each server.	Zero servers are on PHP 7 or 8.1, four are on 8.2 with a scheduled upgrade to 8.4 next month, and the rest run 8.3 or 8.4.
3	Have all our PHP servers been patched to PHP 8.5.12, 8.4.27, 8.3.36 or 8.2.35 for the October 2026 PHP2Shell vulnerabilities, and were worker processes restarted?	Our Linux servers update automatically, we assume we are covered.	Every server was updated and verified within 48 hours of the 21 Oct 2026 advisory, and legacy hosts have interim mitigations and WAF rules applied.
4	Are our servers still broadcasting their PHP version number to Shodan and public scanners in HTTP headers?	We use default PHP settings.	expose_php is Off across all hosts and our reverse proxies strip X Powered By and Server version banners.
5	When was the last time an independent vulnerability scanner tested all our PHP applications?	We ran a scan two years ago during an audit.	We run automated scans every month and after every release using Bithost ZScanner.
6	Do we scan our PHP source code and third party Composer libraries for	Developers review code manually when they have time.	Every repository is scanned automatically with Bithost CodeSecurity and we

No.	Question for the CIO, CTO or IT Head	Red flag answer	Healthy, mature answer
	known vulnerabilities and hardcoded secrets?		maintain an up to date CycloneDX SBOM.
7	If one of our public PHP websites were compromised tonight, could the attacker reach our internal corporate network or customer database?	The web server and database run on the same machine with shared credentials.	Web workers run in isolated pools with least privilege database accounts and strict network segmentation.
8	What is our Legacy Technology Risk Score, and how much budget is allocated to retiring technical debt before branches hit end of life?	We only upgrade when something breaks.	We benchmark our estate with Bithost Legacy Meter and budget annual runtime upgrades six months ahead of every EOL date.
9	Are we ready to meet the 6 hour CERT In incident reporting window and demonstrate reasonable security safeguards under DPDPA 2023?	We would figure out who to notify if an incident happened.	Our logs are NTP synchronised and retained for 180 days, our playbook is tested, and our posture is validated through Bithost Certification Authority.
10	Where we lack internal bandwidth to upgrade old PHP applications, do we have an engineering partner engaged to migrate them safely?	We are waiting until our product roadmap slows down next year.	We have engaged Bithost to audit, refactor and upgrade our legacy PHP applications using a zero downtime blue green rollout.

Table 28.1: Ten board level governance questions on PHP exposure and lifecycle management

Chapter 29. How Bithost Helps Organisations Scan, Upgrade and Protect Their PHP Estate

Everything described so far in this report leads to a practical question. If your organisation has five, fifty or five hundred web applications built over the last decade, how do you find every exposed PHP runtime, test it for vulnerabilities, upgrade legacy code without breaking business operations, and prove to your board, customers and regulators that the job was done properly?

Bithost built an integrated set of security products and engineering services specifically for that journey. You can use our tools in house with your own engineers, or ask Bithost's specialists to run the discovery, scanning, code migration and certification for you end to end.

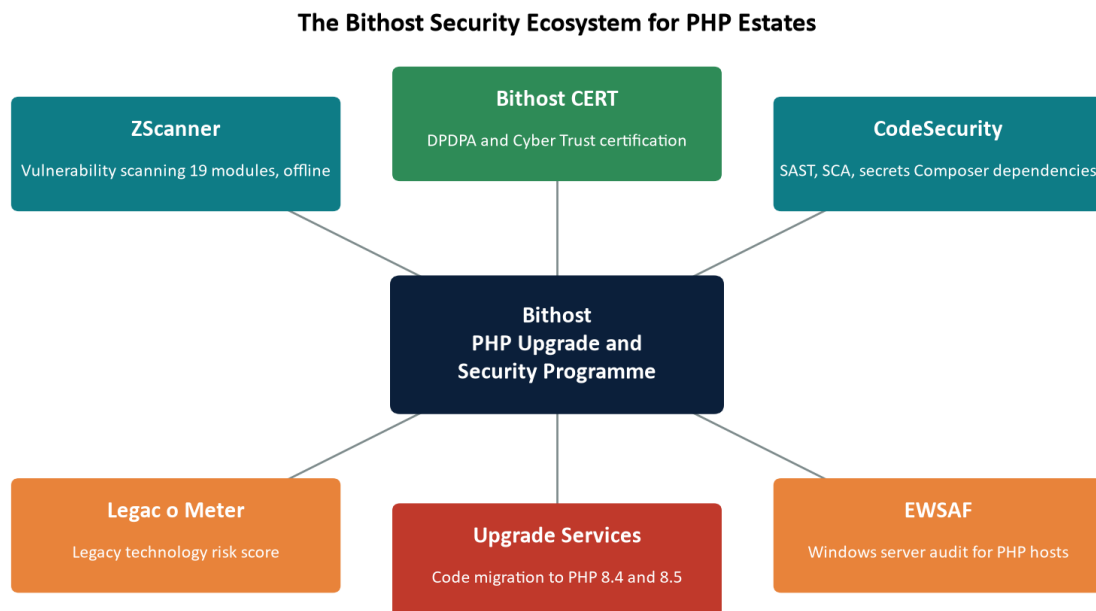


Figure 29.1: How the Bithost product and service ecosystem covers the full PHP lifecycle

29.1 Step 1: Measure your legacy exposure with Legac o Meter

Before writing a single line of code or requesting budget from leadership, technology managers need a clear, objective baseline of how much technical debt and legacy risk the organisation is carrying. Bithost's Legac o Meter, available free at legacometer.bithost.in, is a 25 question diagnostic built for CTOs, CISOs and IT heads.

- Evaluates runtime currency, framework age, dependency hygiene, deployment automation and patching speed in under ten minutes.
- Calculates a weighted legacy risk score and assigns an easy to understand letter grade from A to F.

- Generates a clean, board ready PDF executive report that explains technical debt in business and financial terms so leadership can approve upgrade budgets rapidly.

29.2 Step 2: Scan your live websites and APIs with ZScanner

Once you are ready to inspect your live web applications, Bithost's flagship vulnerability scanner ZScanner (zscanner.bithost.in) gives your team an immediate, deep view of what an attacker sees from the outside and from authenticated sessions.

ZScanner capability	Details	Why it matters for PHP websites
19 dedicated scan modules	Covers more than 40 vulnerability classes including SQL injection, XSS, SSRF, RCE, path traversal, insecure headers, exposed backups, CORS misconfigurations and SSL/TLS weaknesses	Detects both core PHP exposure headers and the application bugs that pile up on older PHP sites
100 percent offline desktop operation	Runs locally on Windows, macOS or Linux with no cloud data upload and no external telemetry	Safe to run against sensitive internal portals, healthcare apps, banking staging environments and air gapped networks
Authenticated and API scanning	Supports cookie, bearer token, session and custom header authentication for full post login coverage	Tests admin dashboards, logged in customer portals and REST/GraphQL backends built in Laravel or Symfony
Direct compliance mapping	Automatically maps every finding to OWASP Top 10 2021, OWASP API Security Top 10 2023 and PCI DSS 4.0	Saves days of manual spreadsheet mapping when preparing for customer audits or payment card assessments
Actionable developer reports	Exports clean HTML, PDF and JSON reports with exact HTTP request/response evidence and remediation steps	Developers can reproduce and fix issues immediately without deciphering noisy false positives

Table 29.1: How ZScanner tests and validates PHP web applications

During a PHP upgrade project, teams run ZScanner twice: once at the start to find every exposed header, forgotten phpinfo file, backup archive and injectable endpoint, and once after the upgrade to verify that hardening rules are active and no regressions were introduced.

29.3 Step 3: Inspect source code, secrets and Composer packages with Bithost CodeSecurity

External scanning shows what is reachable over HTTP. To find vulnerable libraries, hardcoded credentials and unsafe code patterns inside the repository itself, Bithost built CodeSecurity (bithost.in/codesecurity), a fast, self hosted application security platform.

- Static Application Security Testing (SAST) scans PHP source files for command injection, unsafe unserialize() calls, SQL injection, weak cryptography, raw file inclusion and deprecated constructs.
- Software Composition Analysis (SCA) inspects composer.json and composer.lock to flag every third party package with known CVEs or incompatible PHP version constraints.
- Secret detection scans source trees and Git commit history for hardcoded database passwords, AWS keys, payment gateway secrets and API tokens.
- Infrastructure as Code (IaC) scanning checks Dockerfiles, Kubernetes manifests and Nginx/Apache configs to catch root containers, old base images such as php:7.4 apache and missing security flags.
- Generates standard CycloneDX Software Bills of Materials (SBOM) so your organisation has a complete inventory of every component running in production.

29.4 Step 4: Audit Windows servers and IIS/XAMPP hosts with EWSAF

As CVE 2024 4577 demonstrated, PHP running on Windows servers through IIS, Apache or XAMPP carries distinct operating system and locale risks. For organisations running Windows server estates, Bithost's Enterprise Windows Security Audit Framework, EWSAF (ewsaf.bithost.in), provides an agentless, strictly read only security audit across 60 modules mapped to CIS Benchmarks, NIST, ISO 27001, PCI DSS, HIPAA and SOC 2.

29.5 Step 5: Hands on PHP upgrade and modernisation engineering by Bithost

Many organisations know their PHP version is outdated, but their internal developers are fully booked building product features, or the original agency that wrote the application is long gone. Bithost's engineering team steps in to execute the upgrade safely from start to finish.

Service track	Who it is built for	What Bithost delivers
48 Hour Emergency PHP Shielding & Patch Sprint	Organisations needing immediate protection against PHP2Shell (CVE 2026 109618 / 109619) across live servers	Full asset discovery, patching of 8.2 to 8.5 hosts, official interim hardening on legacy hosts, WAF rule deployment and ZScanner verification
Legacy PHP to PHP 8.4 / 8.5 Turnkey Migration	Applications stuck on PHP 5.6, 7.x, 8.0 or 8.1 (custom PHP, Laravel, Symfony, CodeIgniter, CakePHP)	Complete static analysis, dependency replacement, refactoring of broken functions, staging test suite

Service track	Who it is built for	What Bithost delivers
		execution and zero downtime blue green cutover
CMS & Ecommerce Rescue (WordPress, WooCommerce, Magento, Drupal, Moodle)	Stores, universities and public portals blocked from upgrading by broken plugins or custom themes	Plugin audit, replacement or custom patching of incompatible extensions, theme remediation, database tuning and production PHP 8.4/8.5 upgrade
Containerisation & Hardened Infrastructure Rebuild	Teams moving off fragile shared hosting or aging VMs	Packaging the upgraded PHP application into hardened rootless Docker containers with automated CI/CD patching and monitoring

Table 29.2: Bithost PHP upgrade and remediation service tracks

29.6 Step 6: Prove compliance and earn trust with Bithost Certification Authority

After your applications are upgraded, hardened and verified with ZScanner and CodeSecurity, you should get credit for the work you have done. Through the Bithost Certification Authority (cert.bithost.in), organisations can complete independent readiness assessments and earn verifiable certifications covering India's DPDPA 2023, GDPR, ISO 27001 readiness and the Bithost Cyber Trust mark, backed by a public trust registry that enterprise buyers, partners and regulators can verify online.

How to get started with Bithost today

1. Run a free 10 minute legacy assessment at legacometer.bithost.in.
2. Download ZScanner at zscanner.bithost.in and scan your internet facing PHP domains.
3. Explore CodeSecurity at bithost.in/codesecurity, EWSAF at ewsaf.bithost.in and compliance certification at cert.bithost.in.
4. Contact the Bithost security and engineering team through bithost.in for an emergency PHP2Shell review or a fixed scope PHP 8.4/8.5 upgrade proposal.

29.7 Inside ZScanner's 19 Vulnerability Modules for PHP Websites

Security teams often ask what specific checks Bithost ZScanner (zscanner.bithost.in) runs when pointed at a PHP web application or API. Table 17.3 details all nineteen scan modules built into the ZScanner desktop engine and explains the exact PHP weaknesses each module uncovers.

Module No.	ZScanner module name	What it tests against a PHP target	Compliance mapping target
Module 01	Runtime & Banner Fingerprinting	Detects X Powered By, Server headers, PHPSESSID defaults and end of life PHP 5/7/8.0/8.1 signatures	OWASP A05 / A06, PCI DSS 6.3
Module 02	SQL Injection (SQLi) Engine	Tests error based, boolean blind and time based SQL injection across legacy mysql_*, mysqli and raw PDO queries	OWASP A03, PCI DSS 6.2.4
Module 03	Cross Site Scripting (XSS) Analyzer	Detects reflected, stored and DOM XSS where unescaped echo or print statements render user input	OWASP A03, PCI DSS 6.2.4
Module 04	Remote Code Execution & Command Injection	Probes for OS command injection (exec, system, passthru), eval injection and PHP CGI / runtime RCE vectors	OWASP A03 / A06, PCI DSS 6.2.4
Module 05	Local & Remote File Inclusion (LFI / RFI)	Tests include, require and file_get_contents parameters for path traversal and php://filter or phar:// wrappers	OWASP A01 / A03, PCI DSS 6.2.4
Module 06	Server Side Request Forgery (SSRF)	Checks cURL, Guzzle and file_get_contents URL parameters for internal network and cloud metadata (169.254.169.254) access	OWASP A10, API Security Top 10
Module 07	Exposed Environment & Config Discovery	Hunts for readable /.env, /wp config.php.bak, /app/etc/env.php, /composer.json and /composer.lock files	OWASP A05, PCI DSS 2.2
Module 08	Diagnostic & Admin Panel Finder	Discovers forgotten /phpinfo.php, /info.php, /test.php, /adminer.php, /phpmyadmin and debug toolbars (Telescope, Profiler)	OWASP A05, ISO 27001 A.8.9

Module No.	ZScanner module name	What it tests against a PHP target	Compliance mapping target
Module 09	Backup & Archive Exposure Scanner	Identifies left behind .zip, .tar.gz, .sql, .bak, .swp and .old archives inside the web root	OWASP A01 / A05
Module 10	Insecure Deserialization Check	Detects PHP serialized object strings in cookies and parameters passed to unserialize()	OWASP A08
Module 11	File Upload & Web Shell Vector Check	Verifies whether multipart upload endpoints enforce MIME validation and block .php/.phtml execution	OWASP A04, PCI DSS 6.2.4
Module 12	Session & Cookie Security Auditor	Inspects Set Cookie headers for HttpOnly, Secure, SameSite, prefixing and session fixation resistance	OWASP A07, PCI DSS 4.2
Module 13	Authentication & Brute Force Resilience	Tests login forms, password reset flows and rate limiting on PHP authentication controllers	OWASP A07, API2:2023
Module 14	Broken Access Control & IDOR Tester	Checks numeric and UUID object references in routes (/invoice?id=104) across user privilege levels	OWASP A01, API1:2023
Module 15	HTTP Security Header & CSP Validator	Audits HSTS, Content Security Policy, X Frame Options, Referrer Policy and Permissions Policy headers	OWASP A05, PCI DSS 6.4
Module 16	CORS Policy Misconfiguration Check	Tests whether Access Control Allow Origin reflects arbitrary attacker origins alongside Allow Credentials: true	OWASP A05, API8:2023
Module 17	SSL/TLS Cipher & Certificate Inspector	Verifies TLS 1.2/1.3 enforcement, weak cipher rejection, certificate expiry and chain validity	OWASP A02, PCI DSS 4.2.1
Module 18	CMS & Framework Fingerprint Auditor	Detects vulnerable WordPress,	OWASP A06, PCI DSS 6.3

Module No.	ZScanner module name	What it tests against a PHP target	Compliance mapping
		WooCommerce, Drupal, Joomla, Magento and Laravel exposure artifacts	
Module 19	REST & GraphQL API Security Scanner	Tests mass assignment, excessive data exposure, missing function level auth and improper rate limits on PHP APIs	OWASP API Security Top 10 2023

Table 29.3: Complete list of ZScanner's 19 scan modules and how they protect PHP estates

Chapter 30. Operational Guide: Deploying the Bithost Security Suite Across Your PHP Estate

To make it effortless for technology teams to move from reading this report to securing their servers, this final operational chapter walks through exactly how to run each Bithost product against a real PHP estate in a single week.

30.1 Day 1: Running Legaco Meter across your application portfolio

Open legacometer.bithost.in in your browser. Complete the 25 assessment questions covering your oldest production PHP version, how long ago your framework was updated, whether automated tests exist, how dependencies are tracked and how quickly security patches reach production. Download the generated PDF executive report and use the letter grade and weighted score to rank which applications get upgraded first.

30.2 Day 2: Scanning your live PHP domains with ZScanner

Download and launch Bithost ZScanner from zscanner.bithost.in on a security engineer's workstation. Because ZScanner runs 100 percent offline as a native desktop application, no target URLs, session cookies or scan findings are ever transmitted to external third party servers.

ZScanner workflow step	How to configure it for a PHP website	What to inspect in the output report
1. Target & Scope Setup	Enter your root domain (e.g. https://portal.example.com) and any API or staging subdomains	Verify whether the target responds with X Powered By or Server banners revealing PHP 5.x, 7.x, 8.0 or 8.1
2. Authentication Profile	Configure session cookies, login form credentials or Bearer tokens so the crawler reaches authenticated pages	Ensure post login dashboard controllers, profile upload forms and admin endpoints are included in crawl coverage
3. Module Selection (19 Modules)	Enable all 19 modules including SQLi, XSS, RCE, SSRF, Path Traversal, Backup/Config Discovery, Header Analysis and SSL/TLS	Review High and Critical findings first, paying immediate attention to exposed .env files, phpinfo pages, backup archives and injectable parameters
4. Compliance Mapping Export	Select OWASP Top 10 2021, OWASP API Security Top 10 2023 and PCI DSS 4.0 report views	Hand the technical HTML/JSON report to your PHP developers and the executive PDF summary to your compliance and leadership team

Table 30.1: Four step workflow for scanning PHP estates with Bithost ZScanner

30.3 Day 3: Auditing PHP repositories with Bithost CodeSecurity

Point Bithost CodeSecurity (bithost.in/codesecurity) at your Git repositories or application source folders. In a single self hosted pass, CodeSecurity performs four essential checks on your PHP project:

1. Static Application Security Testing (SAST) flags dangerous calls such as `eval()`, `shell_exec()`, `unserialize()` on request data, raw SQL string concatenation and insecure file upload handling.
2. Software Composition Analysis (SCA) parses `composer.json` and `composer.lock` to list every vulnerable Packagist dependency and identify packages blocking your upgrade to PHP 8.4 or 8.5.
3. Secret Detection catches database credentials, AWS keys, payment gateway secrets and API tokens accidentally committed inside config files or historical Git commits.
4. Infrastructure as Code (IaC) & SBOM Generation checks your Dockerfiles and server manifests for outdated base images and exports a standard CycloneDX SBOM for audit readiness.

30.4 Day 4: Auditing Windows servers with EWSAF

If any of your PHP workloads, databases or Active Directory controllers run on Windows Server, run Bithost EWSAF (ewsaf.bithost.in). Because EWSAF is strictly read only and agentless, it makes zero changes to production registry keys or services while auditing 60 security modules and mapping every gap to CIS, NIST, ISO 27001, PCI DSS, HIPAA and SOC 2.

30.5 Day 5 and beyond: Turnkey PHP upgrade and Bithost Certification

With your Legaco Meter score, ZScanner vulnerability report, CodeSecurity source audit and EWSAF infrastructure assessment in hand, your team has a complete blueprint. You can execute the refactoring and PHP 8.4 / 8.5 upgrade internally using Chapters 18 through 26 of this report, or hand the blueprint to Bithost's engineering team for a turnkey zero downtime migration. Once remediation is verified by a clean ZScanner rescan, complete your compliance journey at cert.bithost.in to earn your verifiable DPDPA 2023, GDPR, ISO 27001 readiness or Bithost Cyber Trust certification.

30.6 Final word from Bithost

Outdated PHP is one of the most common, most destructive and most preventable risks on the modern internet. Unlike nation state zero day attacks that require millions of dollars in custom threat research to stop, the exposure described in this report can be eliminated through disciplined asset discovery, timely runtime upgrades, sensible server hardening and continuous verification.

Whether your organisation runs a single ecommerce store, a university network, a hospital diagnostic portal or a portfolio of enterprise web applications, you do not have to wait for an automated exploit script to test your defences. Use the checklists and configurations in this report today, test your live

applications with ZScanner (zscanner.bithost.in), audit your code and dependencies with CodeSecurity (bithost.in/codesecurity), benchmark your legacy risk at Legaco Meter (legacometer.bithost.in), secure your Windows estates with EWSAF (ewsaf.bithost.in), and certify your compliance readiness through Bithost Certification Authority (cert.bithost.in). When you are ready for hands on assistance to upgrade and secure your PHP estate from end to end, our team at Bithost (bithost.in) is here to help.

Appendix A: 50 Point PHP Security and Upgrade Self Assessment Checklist

Use this fifty point checklist to audit any PHP application in your estate. Score one point for every item marked Yes. A score below 35 indicates that the application requires immediate remediation.

No.	Control area	Verification item	Target state
1	Runtime currency	PHP branch is in active or security support on php.net (8.3, 8.4 or 8.5)	Yes
2	Runtime currency	If on PHP 8.2, a funded migration plan is scheduled before 31 Dec 2026 EOL	Yes
3	PHP2Shell patch	PHP version is at least 8.5.12, 8.4.27, 8.3.36 or 8.2.35	Yes
4	Worker restart	PHP FPM, Apache or container workers were restarted after the latest package update	Yes
5	Banner hiding	expose_php is set to Off in production php.ini	Yes
6	Banner hiding	Web server or reverse proxy strips X Powered By and Server version numbers	Yes
7	Error handling	display_errors and display_startup_errors are set to Off in production	Yes
8	Error handling	log_errors is On and logs write to a non public path outside the document root	Yes
9	Session security	session.upload_progress.enabled is set to 0 unless explicitly required and patched	Yes
10	Session security	session.use_strict_mode and session.use_only_cookies are enabled	Yes
11	Session security	session.cookie_httponly and session.cookie_secure are enabled	Yes

No.	Control area	Verification item	Target state
12	Session security	session.cookie_samesite is set to Lax or Strict	Yes
13	Session security	Session cookie name is changed from default PHPSESSID to a custom __Host prefixed name	Yes
14	Remote files	allow_url_fopen is disabled unless required and allow_url_include is Off	Yes
15	Dangerous calls	exec, passthru, shell_exec, system, proc_open, popen and phpinfo are disabled where not needed	Yes
16	Filesystem	open_basedir restricts PHP file access to the application directory and temp path	Yes
17	Filesystem	Web server document root points to /public or /web subfolder, not the repository root	Yes
18	Filesystem	Application source files and vendor directory are owned by root or deploy user and read only to the PHP worker user	Yes
19	Upload safety	Nginx or Apache explicitly blocks execution of .php, .phtml and .phar files inside upload folders	Yes
20	Upload safety	Uploaded files are validated by MIME content and stored outside the web root or on object storage	Yes
21	Request limits	post_max_size and upload_max_filesize are set to the minimum required by the application	Yes
22	Request limits	max_input_vars, max_input_nesting_level and max_file_uploads are explicitly bounded	Yes
23	Proxy boundary	Reverse proxy enforces client_max_body_size, header	Yes

No.	Control area	Verification item	Target state
		count limits and short request timeouts	
24	Secrets hygiene	.env, wp config.php and config files are unreadable over HTTP and blocked by web server rules	Yes
25	Secrets hygiene	No backup files (.bak, .swp, .old, .sql, .zip) exist inside the public document root	Yes
26	Diagnostic files	No info.php, phpinfo.php, test.php or adminer.php files exist on production servers	Yes
27	Process isolation	Each PHP site runs under its own dedicated Linux user and dedicated PHP FPM pool	Yes
28	Systemd sandbox	PHP FPM service uses PrivateTmp, ProtectSystem=strict and NoNewPrivileges	Yes
29	Windows CGI	PHP CGI mode is disabled on Windows servers in favour of FastCGI with strict character filtering	Yes
30	Database access	All database queries use PDO or mysqli prepared statements with bound parameters	Yes
31	Database access	Database account used by PHP has least privilege (no FILE, SUPER or DROP DATABASE rights)	Yes
32	Serialization	Application never passes untrusted user input to unserialize() or uses allowed_classes=false	Yes
33	Cryptography	Legacy mcrypt and md5/sha1 password hashing are replaced by password_hash() with Argon2id or bcrypt	Yes
34	Dependencies	composer.lock is committed to version control and audited with composer audit or CodeSecurity	Yes

No.	Control area	Verification item	Target state
35	Dependencies	No abandoned Packagist packages with known unpatched CVEs remain in production	Yes
36	CMS hygiene	WordPress, Drupal, Joomla, Magento or Moodle core is on a currently supported release	Yes
37	CMS hygiene	All unused plugins, themes and inactive admin accounts have been removed	Yes
38	Headers	Content Security Policy, HSTS, X Frame Options and X Content Type Options headers are set	Yes
39	WAF protection	A Web Application Firewall inspects inbound traffic for RCE, SQLi, XSS and scanner floods	Yes
40	Container build	Docker images pin explicit patched PHP tags (e.g. 8.4.27) and run as a non root user	Yes
41	Cloud metadata	Cloud instances enforce IMDSv2 (hop limit 1) so SSRF or RCE cannot steal IAM credentials	Yes
42	Egress filtering	Outbound firewall rules block the web server from initiating arbitrary connections to the internet	Yes
43	Logging	Access logs, PHP error logs and FPM slow logs are shipped to a central tamper evident log server	Yes
44	Time sync	Server clocks are synchronised via NTP and logs are retained for at least 180 days (CERT In compliance)	Yes
45	File integrity	File integrity monitoring alerts on any new or modified .php file in the web root	Yes
46	Backups	Database and media backups are encrypted, tested monthly and stored offline or on immutable storage	Yes

No.	Control area	Verification item	Target state
47	Vulnerability scan	Application is scanned monthly and after every release using Bithost ZScanner	Yes
48	Code review	CI pipeline runs static analysis, secret scanning and SCA using Bithost CodeSecurity	Yes
49	Incident plan	Team has a documented 6 hour incident reporting workflow and emergency maintenance playbook	Yes
50	Governance	Every PHP asset has a named owner and an automated calendar reminder 6 months before branch EOL	Yes

Table A.1: 50 point PHP security and upgrade self assessment checklist

Appendix B: Glossary of Technical and Regulatory Terms

Term	Plain English definition
Active Support	The first two years of a PHP branch's life, during which the PHP project fixes both general bugs and security flaws in regular point releases.
Backport	Taking a security fix written for a newer software version and adapting it to apply to an older version number.
CGI (Common Gateway Interface)	An older method for web servers to run external programs such as PHP to generate dynamic pages, largely superseded by FastCGI and PHP FPM.
Composer	The standard dependency manager for PHP, used to install and update third party libraries recorded in composer.json and composer.lock.
CVE (Common Vulnerabilities and Exposures)	A globally standard reference identifier assigned to a publicly known cybersecurity vulnerability, such as CVE 2026 109618.
CVSS (Common Vulnerability Scoring System)	A numerical scale from 0.0 to 10.0 used to rate the severity of software vulnerabilities. Scores of 9.0 and above are rated Critical.
CycloneDX SBOM	A standard machine readable Software Bill of Materials format that lists every library and component inside an application.
DPDPA 2023	India's Digital Personal Data Protection Act 2023, which requires Data Fiduciaries to maintain reasonable security safeguards and imposes penalties up to INR 250 crore.
End of Life (EOL)	The date after which a software branch no longer receives bug fixes or security patches from its upstream maintainers.
expose_php	A php.ini configuration setting that, when left On, advertises the server's exact PHP version in the X Powered By HTTP header.
FrankenPHP	A modern application server built on Caddy that embeds the PHP interpreter for standard and persistent worker execution.
mod_php	An Apache web server module that embeds the PHP interpreter directly inside each Apache worker process.
Out of Memory (OOM)	A state where a program exceeds its allowed memory allocation. In CVE 2026 109618, OOM handling during partial initialization can corrupt cleanup pointers.

Term	Plain English definition
OPcache	PHP's built in bytecode cache, which stores precompiled script bytecode in shared memory so PHP does not have to parse files on every request.
PCI DSS 4.0	The Payment Card Industry Data Security Standard version 4.0, governing any organisation that stores, processes or transmits cardholder data.
PHP FPM (FastCGI Process Manager)	The standard modern way to run PHP in production, where a pool of persistent worker processes handles requests passed from Nginx or Apache.
PHP2Shell	The industry name given to CVE 2026 109618 and CVE 2026 109619, two critical unauthenticated remote code execution vulnerabilities disclosed on 21 October 2026.
Remote Code Execution (RCE)	A vulnerability that allows an attacker to run arbitrary commands or code on a target server over a network.
SAPI (Server Application Programming Interface)	The interface layer connecting the PHP engine to a specific web server or command line environment.
SAST and SCA	Static Application Security Testing (scanning your own source code) and Software Composition Analysis (scanning third party packages and dependencies).
Security Fixes Only	Years three and four of a PHP branch's lifecycle, when only critical security vulnerabilities receive upstream patches.
Web Shell	A malicious script uploaded or injected onto a web server that gives an attacker a persistent backdoor to run commands, browse files and exfiltrate data.

Table B.1: Plain English glossary of key terms

Appendix C: Primary Sources and References

1. PHP Group. Security advisory: Critical remote code execution vulnerabilities, update now (CVE 2026 109618 / GHSA w8hw w9fx j79w and CVE 2026 109619 / GHSA 6wmf 4rv5 p3rp), published 21 October 2026 on php.net, fixing releases PHP 8.5.12, 8.4.27, 8.3.36 and 8.2.35.
2. PHP Group. Informational advisory: Session upload progress (referencing CVE 2026 109619 mitigation session.upload_progress.enabled=0 and upcoming regular release advisory CVE 2026 115560), published 23 October 2026 on php.net.
3. PHP Group. Supported Versions and End of Life Branches tables, php.net/supported versions.php and php.net/eol.php, accessed 24 October 2026.
4. W3Techs (Q Success). Usage statistics and market share of PHP for websites, October 2026 survey data as publicly reported.
5. Censys, OX Security and Black Kite internet exposure and Shodan telemetry analyses following the October 2026 PHP2Shell disclosure (36.7 million service records, 1.9 to 2.2 million measurable PHP hosts, 1.37 million on legacy/vulnerable configurations, 99.3 percent unpatched at 24 hours).
6. National Vulnerability Database and CISA Known Exploited Vulnerabilities entries for CVE 2024 4577 (PHP CGI Windows Best Fit argument injection, CVSS 9.8) and threat telemetry from GreyNoise, Akamai and Bitdefender.
7. IBM and Ponemon Institute. Cost of a Data Breach Report 2025 (Global average USD 4.44 million, United States USD 10.22 million, India INR 220 million / USD 2.51 million, Healthcare USD 7.42 million).
8. Sophos. The State of Ransomware 2025 (3,400 respondents, exploited vulnerabilities as top root cause at 32 percent, mean recovery cost USD 1.53 million).
9. Verizon. 2025 Data Breach Investigations Report (DBIR) (Vulnerability exploitation as initial access vector at 20 percent overall, stolen credentials in basic web application breaches at 88 percent).
10. Government of India. The Digital Personal Data Protection Act, 2023 (No. 22 of 2023), Section 8(5) and Schedule of Penalties, and Indian Computer Emergency Response Team (CERT In) Directions under Section 70B(6) of the Information Technology Act, 2000, dated 28 April 2022.
11. Bithost Security Ecosystem documentation: ZScanner (zscanner.bithost.in), CodeSecurity (bithost.in/codesecurity), EWSAF (ewsaf.bithost.in), Legaco Meter (legacometer.bithost.in) and Bithost Certification Authority (cert.bithost.in).