

Designing Infrastructure for the AI Era

The AI that helps businesses find their weaknesses also helps attackers. Four design decisions limit what one compromised application can reach and how quickly the business recovers.



THE BRIEF

AI helps researchers find software weaknesses. Those tools are also available to attackers. On 21 October 2026, the maintainers of PHP, the programming language behind roughly 70% of websites whose server-side language is known, released fixes for two critical flaws found with AI assistance. The flaws had survived in its code for more than a decade and could let an outsider take control of an affected application without logging in. The first finding came two days into an audit; the public fix followed ten weeks after the first report. Four days before release, Amazon detected an attempt to exploit them against a decoy system. AI can help both sides move faster, but repairing every affected system still takes time. Leaders must limit what one compromised application can reach and rehearse recovery, so an unavailable patch need not leave the business choosing between broad exposure and shutting everything down. ^{1, 2, 3, 4, 5}

IN THIS ARTICLE

01	A Ten-Year-Old Flaw, Found in Days and Fixed in Weeks	03
02	The Same Tools on Both Sides	04
03	How One Compromised Application Opens a Route to Others	06
04	Four Design Decisions That Limit the Damage	08
05	1. Separate Public Information from Sensitive Functions	08
06	2. Restrict What an Attacker Can Do Inside an Application	09
07	3. Limit Access to the Data and Actions Each Task Requires	11
08	4. Prepare to Repair and Recover	12
09	Where to Invest First	13
10	Scenario A and Scenario B: The Same Flaw, Two Designs	14
11	Three Decisions for the Board This Quarter	16
12	Outlook	17
—	Key takeaways	18

01

A Ten-Year-Old Flaw, Found in Days and Fixed in Weeks

Developers use AI to write code, and security researchers use it to read that code and point out what could go wrong. The second use is the one that matters for this article, because a tool that can show a defender where a system is weak can show others the same thing.

In the summer of 2026, the security firm Calif tested how far that capability had come. On 30 July, its researchers began examining PHP, a programming language used to build websites and business applications. WordPress, the most widely used website publishing system, runs on it. W3Techs found PHP on 69.3% of websites whose server-side language it could identify on 23 October. That measures adoption, not the number of vulnerable websites, but explains why a serious flaw in PHP concerns so many organisations at once.^{6, 7, 1}

Calif did not expect to find much. PHP had been examined for years and, as the firm wrote afterwards, "everyone and every lab had already done extensive AI assisted audits of this target." Yet on 1 August, two days into the work, its researchers found the first of the flaws that became known as PHP2Shell. They combined generally available AI models with their knowledge of earlier PHP failures. The weakness sat in the PHP runtime, the software that executes applications on a server, in code more than a decade old. Under the right conditions, an outsider could make it run the attacker's own code without an account.^{2, 3}

Other researchers soon joined. Wiz, a cloud-security company, found further routes to the same result and used AI to develop working exploits: programs demonstrating how a weakness could be used to break in. Wiz says that "a single researcher completed in a week what previously would have taken a world-class team months of work." This is the company's assessment, not a controlled comparison. Its account shows AI assisting both discovery and the development of an attack.³

Repair was slower. Calif reported its first finding to the PHP project on 11 August; the public release followed on 21 October, ten weeks later. The PHP Foundation says signs of the underlying problem had appeared after an audit in 2024. Proposed fixes risked breaking existing applications or creating other security problems, and without a demonstrated attack the urgency was difficult to establish. Limited engineering capacity and a consensus-based decision process contributed to repeated delays. Meanwhile, the team was assessing more reports, including over a hundred since July that proved invalid, duplicated or impossible to reproduce.⁵

While the fix was being prepared, someone else was testing the flaws. On 17 October, Amazon detected an attempt against MadPot, its network of decoy servers. How the attacker learned of the weaknesses is unknown, and Amazon could not say whether the attempt would have succeeded against a real system. On 20 October, the vulnerability identifiers began circulating early, prompting the PHP project to bring its release forward by a day; the Foundation says no technical details leaked at that stage. After disclosure, attempts multiplied. Fortune reported tens of thousands of attempts within 45 minutes, citing a post by Matthew Prince, Co-founder and Chief Executive Officer of Cloudflare. ^{4, 5, 8}

Attempts do not establish successful attacks. The reporting reviewed contained no confirmed live-system compromise as of 23 October. Operators still faced a decision: the UK National Cyber Security Centre advised updating, applying temporary protections where that was not possible, and considering suspension if neither could be done promptly. Germany's digital ministry suspended websites as a precaution, Al Jazeera and Reuters reported. Dutch government services including DigiD were also suspended, according to the broadcaster NOS. Even without a confirmed breach, precautions can interrupt services. Separating functions in advance gives a business more choice over what it takes offline. ^{9, 10, 11}

02

The Same Tools on Both Sides

Whoever was probing PHP2Shell, none of the sources reviewed establishes that they used AI. The relevance is that the capability researchers used is available to others. The PHP Foundation expects these tools to accelerate discovery for defenders and attackers alike. ⁵

Some attackers are already using them. In November 2025, Anthropic reported that a group it assessed as state-sponsored had used Claude Code for an estimated 80 to 90% of an espionage campaign against about thirty targets, succeeding in a small number of cases. These are the vendor's estimates. That month, Google's Threat Intelligence Group also reported malware that consulted a language model during live operations. These accounts show AI assisting intrusions as well as defensive research. ^{12, 13}

That raises a separate question: are vulnerabilities found with AI being exploited more often than others? VulnCheck's analysis of the first half of 2026 found known exploitation of 14 out of 1,061 AI-discovered

weaknesses, roughly 1.3%, close to its overall comparison rate. Later attacks and delayed detection may change that picture. Mandiant, Google's incident-response business, separately states that it does not consider 2025 the year in which breaches were the direct result of AI. Neither observation measures every way attackers use AI.^{14, 15}

Defenders can use AI for repair too: the PHP Foundation describes assistance with assessing reports, and Wiz with remediation. But finding a weakness does not put a dependable repair into every affected system. Engineering, testing and coordination still take time. This gap predates AI: Mandiant estimates that in 2025, exploitation began on average seven days before a fix was available. That aggregate is not a timetable for every flaw. It does show why prompt updating alone cannot eliminate exposure.^{5, 3, 15}

Repair also depends on suppliers. Some network-security products contain PHP, and on 23 October Palo Alto Networks' advisory listed PAN-OS fixes as forthcoming and gave no fix date for Panorama, its management product. Customers could not install a fix that did not yet exist. An attacker needs one exploitable system; a defender must account for each exposed dependency.¹⁶

Our judgement is that broader access to flaw-finding tools makes it more urgent to prepare for this situation: an exploitable component that cannot yet be repaired. What can it reach while the business waits? AI changes the priority of answering that question, rather than the principles behind the answer.

Assumption

What the evidence means for a decision

Long use and previous audits mean the serious flaws have been found

PHP had been audited extensively, including with AI, and critical flaws remained

Installing updates promptly prevents compromise

Updates are essential, but attacks can begin before an update exists

AI-assisted discovery has already produced a surge in successful attacks

Attackers use AI, but VulnCheck's data does not show AI-found flaws being exploited more often

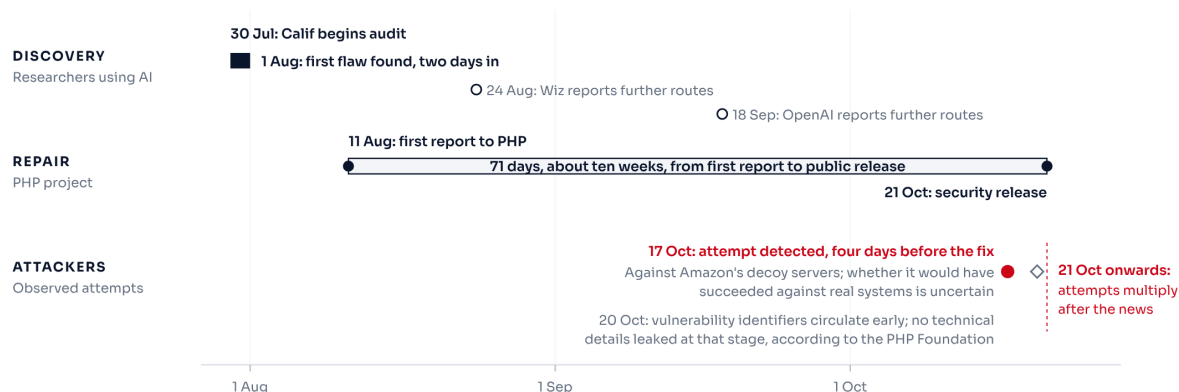
Our existing separation and access limits would contain a compromise

Test these established controls against the dependencies and permissions a compromised application would retain

EXHIBIT 1 · PHP2SHELL

Found in days, fixed in weeks

How the discovery of PHP2Shell, its repair and the first attempts to exploit it unfolded in 2026.



Sources: Calif; Wiz Research; The PHP Foundation (21 Oct 2026); Amazon Threat Intelligence (21 Oct 2026). Attempts are not confirmed compromises; the reporting reviewed contained no confirmed compromise of a live system as of 23 October 2026. 71 days is our calculation from the published dates.

03

How One Compromised Application Opens a Route to Others

To see why reach matters, consider an ordinary customer portal, the kind many companies run so that clients can track orders and download documents. To do that job, the portal has to read from the customer database and ask other systems for order information. An attacker who takes control of the portal inherits those permissions. If the portal can read every customer record, so can the attacker.

There may be little time to intervene. In the hand-offs Mandiant observed, the median interval between an initial intrusion and its transfer to another attacker group fell from more than eight hours in 2022 to 22 seconds in 2025. A business cannot assume it will detect the first intrusion before others exploit it. The permissions already in place therefore matter. ¹⁵

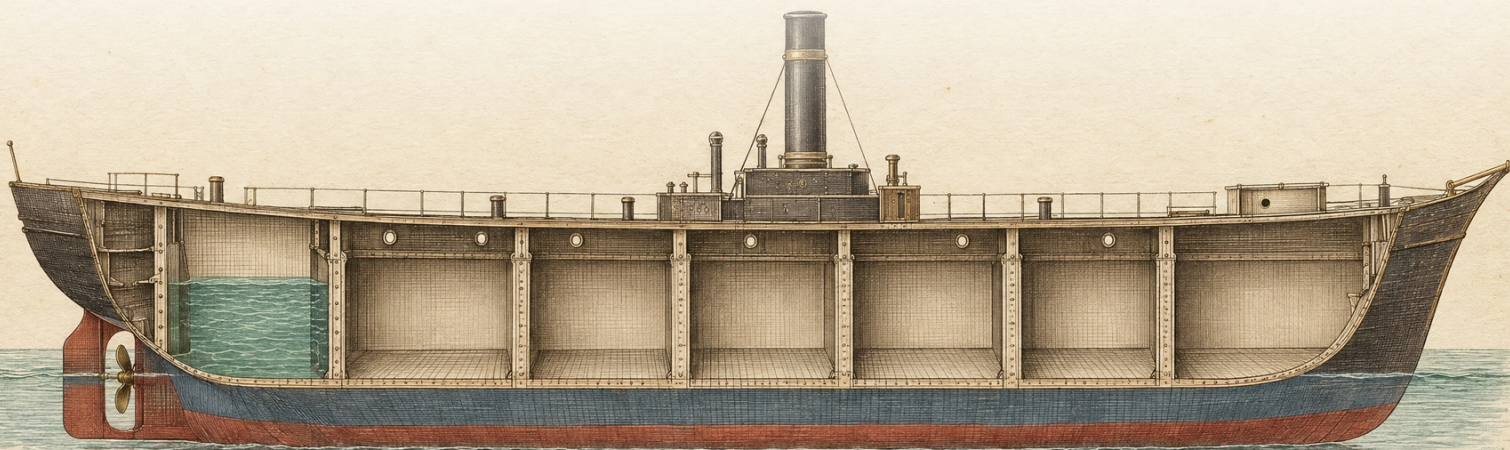
The same pattern appears well beyond websites. Drift, a chat product, connected to its customers' Salesforce systems using credentials, the digital keys that let one service act inside another. In 2025, attackers stole those keys and used Drift's legitimate connections to extract customer records in bulk. Investigators at Unit 42 found the attackers searching the stolen data for further credentials, looking for the next system to enter. ¹⁷

The route can also start in the tools that build software. Developers assemble applications from shared packages downloaded from public libraries such as npm. In November 2025, a campaign called Shai-Hulud 2.0 used hijacked maintainer accounts to publish malicious versions of such packages. When developers installed them, the code ran on their machines and build systems and collected the credentials stored there. The cited investigations do not attribute either incident to AI. Both show how far access granted for one purpose can carry an attacker.¹⁸

AI agents bring the two sides of this story together. Businesses give them access to email, documents and internal tools to work on employees' behalf. But a document or message can contain instructions intended to redirect the agent, a technique called prompt injection. If the agent can export records or change permissions, a malicious message may turn that authority against the company. OWASP's application-security guidance identifies both prompt injection and excessive agency, meaning more authority than a task requires, as principal risks.¹⁹

Each case shows how access that already exists can magnify an intrusion. How much access an application holds is a design decision, and businesses can make it long before the next flaw is found.

Bulkheads let a ship survive one flooded compartment. Separation does the same for a business.



04

Four Design Decisions That Limit the Damage

Back at the customer portal, four such decisions follow the attacker's path: reaching it, taking control, using its access and remaining there. Each offers a practical starting point and a check on the result. They complement prompt updating by limiting what is at stake before an update is available or installed. These are established principles; the work is to make them effective in the systems a business already runs.

05

1. Separate Public Information from Sensitive Functions

A customer login can protect accounts without protecting software that handles requests before login. PHP2Shell exposes that distinction. If product pages, login and order tracking share one application, visitors reach the same underlying software and switching it off removes all three functions. That may be a reasonable design, but the shared dependency has consequences.

Pages needing no live processing can instead be published as static files: finished pages delivered directly, often through a content delivery network that distributes copies. Their requests no longer pass through the application, and product information can stay online while the portal is suspended. This removes unnecessary processing and can preserve continuity. The portal itself remains publicly reachable and still needs patching and protection.

Start by identifying pages that could be published independently, then check their remaining dependencies, such as a shared publishing system, server or identity service. In a test environment, switch the portal off and confirm that the public pages still work. Different web addresses alone prove neither separation nor independence.

Staff administration needs a separate check. An access service can verify identity and, where appropriate, device security before a request reaches the administration software. Teams must also close alternative routes, such as the same administration pages remaining accessible through the portal's public address.

Application and infrastructure teams, or the hosting provider, should demonstrate both results: public pages surviving a portal shutdown and administration accessible only through the access service. On a suitable platform, this may be configuration work; in a tightly coupled application it can require substantial redevelopment. Public content needs a publishing process, personalised functions still need live processing, and staff access depends on the identity service.

06

2. Restrict What an Attacker Can Do Inside an Application

Customers still need to reach the portal, so the second decision concerns what happens if it is compromised. Limited operating-system permissions restrict which files and programs it can change. Read-only program files make persistent alterations harder, while a separate storage area holds what the application needs to write.

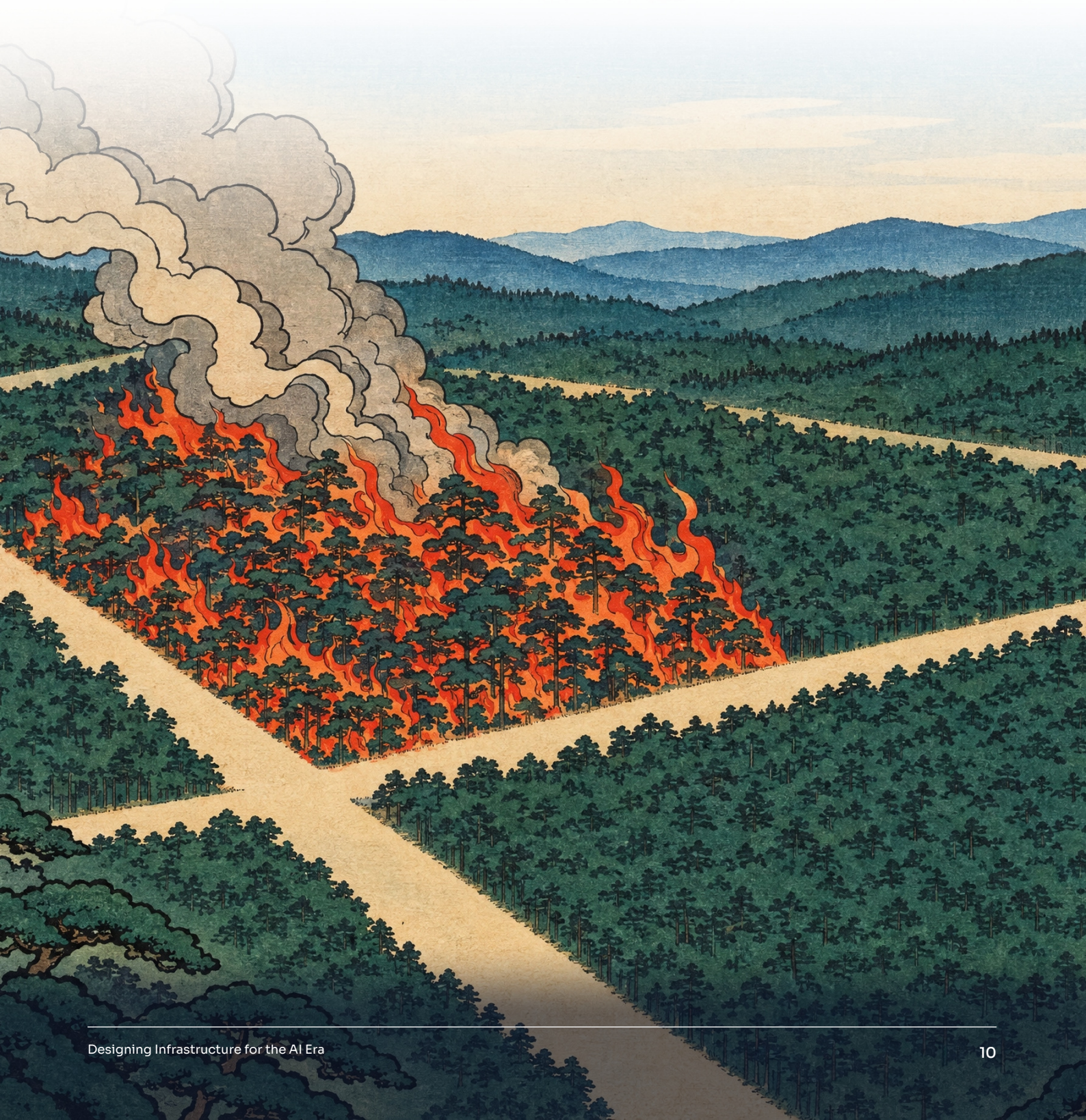
PHP2Shell shows why configuration matters. PHP-FPM, a common way of running PHP applications, ends a worker process when request preparation fails. This blocked some demonstrated attack routes, but others worked after successful preparation. It reduced exposure without removing the need to update. Containers, which give applications separate operating environments, likewise require careful configuration; their presence alone does not establish effective isolation.^{20, 21}

Network access needs the same discipline. A portal may need its database and order-status service; it does not need every internal system. Block connections by default and allow the necessary ones, in both directions. Kubernetes, a platform for running applications, recommends this and warns that its network components must actually enforce the rules. The principle also applies through suitable cloud network controls or firewalls; adopting Kubernetes is not a prerequisite.²²

To avoid breaking a live service, application owners, infrastructure teams and suppliers must map its dependencies together, including supporting services such as logging, backups and email. Observed traffic helps, but can miss month-end or annual processes. Test restrictions before introducing them in production, with monitoring and a way to reverse a faulty change. Enforce the rules outside the application, without giving its identity permission to change them.

Then test the boundary from inside the portal's environment: can it still track an order while being denied a connection to an unrelated internal system? This authorised check tests enforcement, not just configuration. Permitted connections remain a risk, however; stolen information can leave through ordinary responses. The next decision therefore concerns what those connections allow.

Firebreaks do not stop a fire starting; they decide how far it spreads.



07

3. Limit Access to the Data and Actions Each Task Requires

Reaching the order system should not grant access to every order. A service between the portal and the database can return only the fields needed for a customer's order status. It must independently verify both identity and permission, rather than trusting a customer number the compromised portal supplies. One design uses proof issued by an identity service that the portal cannot forge and the order service can verify. The portal must also have no alternative route to read the order database directly. These changes limit blanket access, but valid customer sessions and any other permissions available to the portal remain exposed.²³

Each application also needs its own identity and only the permissions its job requires. That allows more selective restriction or revocation than a shared account. Where supported, short-lived credentials reduce the period during which a stolen copy remains usable.

Expiry is not enough while the application remains compromised: it may keep requesting fresh credentials, as Kubernetes applications can do through automatic renewal. Teams must therefore rehearse withdrawing the application's authority altogether. Name who can do it, measure how quickly, and check which business functions stop. Separate identities make revocation more selective, not disruption-free.²⁴

Purchased services depend on what suppliers support. Before connecting a chat or marketing integration to customer data, establish which permissions can be limited. Drift illustrates what stolen keys can do with existing access. If permissions cannot be narrowed sufficiently, options include connecting less data, choosing another product or explicitly accepting the documented exposure.

For AI agents, enforce limits in the connected systems; asking a model to behave safely is not an access control. A tool reading invoices need not approve payments. Require meaningful approval for consequential actions: the reviewer must see the actual action, affected records and recipient, and the tool must enforce the decision. Routine, lower-risk work can proceed under narrow automatic permissions. High volume alone does not justify removing approval from high-impact actions; redesign the workflow if meaningful review becomes impractical.²⁵

Architects and business owners define the necessary access; procurement checks supplier support. Older applications may require permanent passwords, and narrower interfaces take engineering effort. Those costs

must be weighed against the data and operations exposed by broader authority.

08

4. Prepare to Repair and Recover

The first three decisions limit what an intruder can do while the flaw remains open. The fourth addresses installing a fix and recovering after possible compromise. A software image, a defined package of application code and its required components, gives engineers a repeatable basis for rebuilding with corrected PHP, testing and restoring service. External services and data still need their own recovery arrangements.

If an attacker may have entered, the team must also investigate, withdraw exposed access, check for altered data and restore trustworthy records where needed. Rebuilding code does not undo theft or repair the database.

Ask the team or provider to demonstrate recovery in a controlled exercise: rebuild the service, restore its data, complete a customer task and measure the time against the business's tolerance. Include deployment tools, credentials, backups and supplier support in the exercise. It cannot prove recovery from every attack, but it tests dependencies and replaces an assumed recovery time with a measurement.

That work requires an inventory. A software bill of materials lists components; teams must connect it to deployed versions, service owners and repair responsibilities. Keep critical components on supported versions to reduce the risk of needing a major upgrade before a security fix can be installed.

For embedded software in appliances, the manufacturer must supply a supported fix and the customer or provider must install it. While waiting, follow the vendor's mitigation guidance. Palo Alto Networks, for example, identifies the PAN-OS management web interface as the affected interface, making restrictions on management access relevant. Before purchase, establish who supplies and installs fixes and what can safely remain available during the wait. ¹⁶

Prepare to run with less while repair is pending. The NCSC's advice to consider suspension is easier to act on when one vulnerable function can stop independently. Test that separation, including shared services: a public website intended to survive a portal shutdown must not quietly depend on it.

For new development or planned renewal, consider memory-safe languages, which prevent a large class of memory-handling errors. Google's Android experience supports reducing these vulnerabilities at source. This is a longer-term design choice with migration and compatibility costs, rather than an emergency instruction to rewrite existing systems.²⁶

Engineering teams own recovery for the layers they operate; suppliers own theirs. Both must make compatibility testing and recovery a maintained capability, with responsibilities agreed before an emergency.

09

Where to Invest First

Begin with systems whose compromise or outage would hurt most. Then ask how easily each can be reached, how much access it holds and how long repair would take. Use those answers to prioritise work; they are not a calibrated risk score.

A broadly privileged portal that is difficult to upgrade deserves attention, as does a network appliance dependent on a manufacturer's fix. Include agents and build systems: documents or packages can reach them without a public login page. A system with little access may also be critical if its outage stops sales or a safety-related operation.

For each priority service, require a short account of its exposure, the next worthwhile change, its owner, dependencies and likely effort, and the evidence that will demonstrate improvement. Removing unnecessary public exposure may be an early gain. Where exposure is necessary, narrower permissions and connections can reduce risk while recovery improves. Monitoring remains essential because permitted access can still be misused.

Smaller firms can commission this work from their provider or a qualified external specialist; larger firms can use internal engineering and business owners. Begin with one critical service and a scoped exercise rather than assuming an in-house security team. A platform change or redesign should follow from a specific missing capability and an estimate of the work required.

EXHIBIT 2 · WHERE TO INVEST FIRST

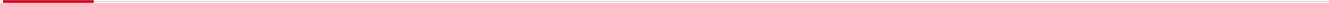
Start where reach, access and repair time combine

A conceptual guide to prioritising design work. Individual systems vary; this is not a measured ranking.

SYSTEM	HOW IT CAN BE REACHED	WHAT AN ATTACKER GAINS	WHAT REPAIR DEPENDS ON	FIRST DESIGN QUESTION
Customer portal with broad database access	Directly from the internet by any visitor	Whatever its permissions allow, such as the whole customer database	Your team, plus the software vendor confirming compatibility	Which records and systems does it actually need?
Network appliance running embedded software	Through its management or network interfaces	A privileged position on the company network	Manufacturer supplies the fix; customer or provider tests and installs it	What can be restricted or switched off while we wait?
AI agent or software-build system holding credentials	Through the documents, messages or packages it processes; no public login needed	The credentials and tools it holds, and the systems they open	Removing the cause, revoking exposed access and restoring a trusted environment	What access does the task need, and how can it be withdrawn?
Static public website	Publicly, but no application runs for each visitor	Ability to alter public content; further reach depends on hosting and publishing access	Securing hosting and publishing access, then restoring trusted content	Is the publishing system kept off the public path?

ACROSS EVERY ROW: BUSINESS CONSEQUENCE
Start with the systems whose failure would hurt most. Even a system with little access can be a priority if its outage stops sales or a safety-related operation.

Editorial framework based on the article's analysis; not measured results or a scoring model.



10 Scenario A and Scenario B: The Same Flaw, Two Designs

The following comparison is illustrative; it describes no specific company or measured result.

A mid-sized industrial supplier runs a PHP customer portal where clients track orders and download product certificates. Its public website describes its products. On the morning PHP2Shell is announced, its IT team learns that the portal is affected and that the portal's vendor has not yet confirmed whether the update is compatible. The team has to decide what stays online while it waits.

Scenario A: the status quo. The public pages, customer portal and staff administration share one PHP application on one server. The portal uses a stored password that gives it access to the entire customer database, and

the server can connect freely to the company's internal systems, including the software that runs its operations. The team faces a poor choice. Taking the server offline removes the public website along with the portal. Keeping it running leaves an attacker who succeeds with broad access to customer data and a path into the rest of the company.

Scenario B: the four decisions applied. Public pages are served independently, and staff administration runs separately behind its own access check. The portal runs with restricted permissions and read-only program files. Network rules outside its control permit its database, identity and order-status services, plus explicitly approved supporting connections. Database credentials are short-lived and limited to portal records. The order service independently checks the customer's identity and permission for each request. The team has tested these boundaries.

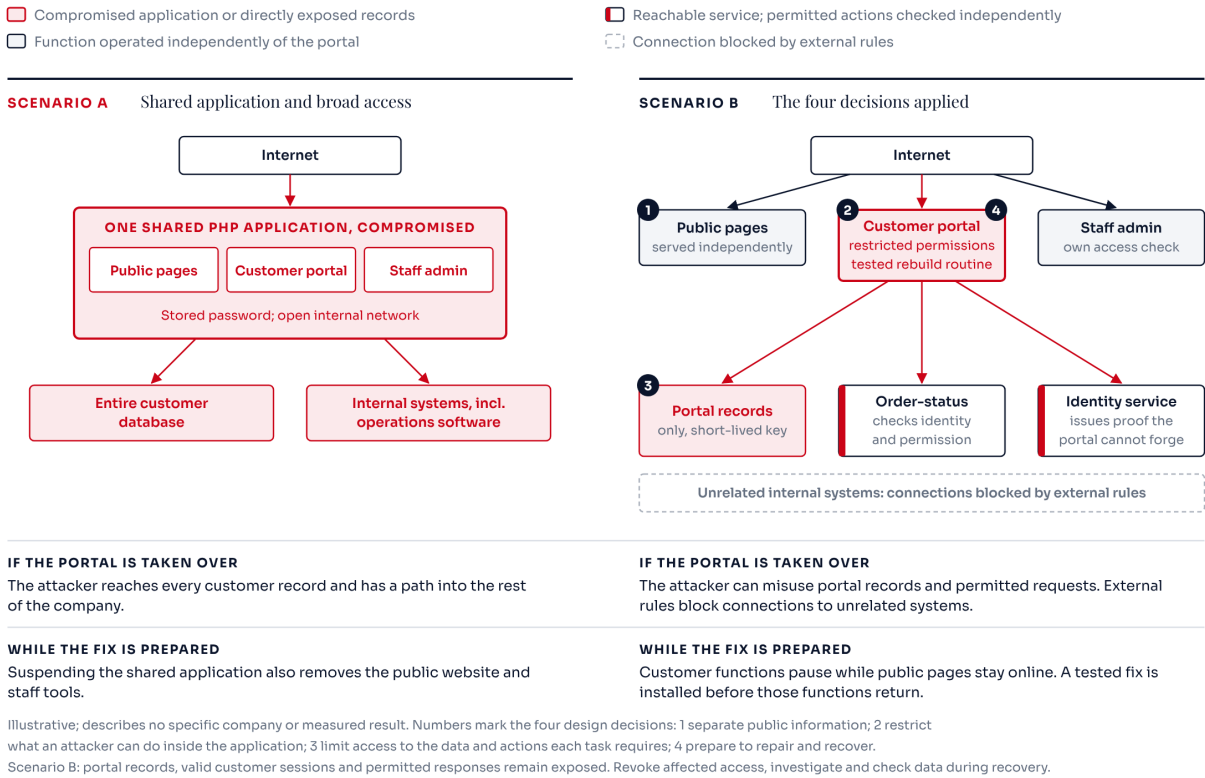
This team can suspend order tracking and certificate downloads while keeping public information online. It uses a rehearsed process to build and test the corrected portal, restoring customer functions once compatibility is established. The team still needs a usable fix; what it has gained is a tested way to limit the interruption and restore service.

If an attacker has already entered, reachable records, valid customer sessions and permitted responses remain at risk. The team must withdraw exposed access, investigate and check the data during recovery. The architecture is intended to restrict the intrusion and preserve independent functions; it does not establish that the compromise stayed inside the portal.

EXHIBIT 3 · ILLUSTRATIVE SCENARIO

One breach, two designs

The same PHP flaw in a supplier's customer portal. What an attacker who takes over the portal can reach, and what the business can keep running while it repairs it.



Three Decisions for the Board This Quarter

01 Commission a ranked map of critical systems.

For each, identify how it can be reached, what data and operations it can access, how long repair would take and what that depends on, including suppliers. Include connected services, appliances, AI agents and software-building systems, and name an owner for each.

02 Fund specific changes for the highest-priority systems, and ask for proof.

Agree which unnecessary connections and permissions will be removed and which business functions must keep running if a service is suspended. Expect evidence back, not assurances: an authorised test showing that unrelated systems cannot be reached, a rehearsed withdrawal of an application's access, and a recovery exercise with a measured time.

03 Make these requirements a condition of new investment.

Before any system, integration or AI agent is bought or built, require a clear account of what its compromise would expose, how its access can be withdrawn and how it can be repaired. Treat broad, permanent access as an exception that needs a business justification.

12

Outlook

PHP2Shell was found by researchers who reported it responsibly. The same tools may help others find weaknesses they choose not to report. AI can assist repair too, but businesses must still coordinate engineers, suppliers and testing before a fix is safely in place.

Leaders cannot choose when the next flaw appears. They can decide what each application can reach, which functions depend on it and how recovery will work. Making and testing those decisions in advance gives the business more options when a warning arrives, with less to redesign under pressure.

KEY TAKEAWAYS

The decisions that endure.

- **Prepare for periods when patching is not yet possible.**

AI assists attackers and defenders; dependable repairs still take engineering and coordination. Limit the consequences while that work proceeds.

- **Limit what one compromise can reach, and test the limits.**

Separate functions where it reduces exposure or preserves service, enforce permissions outside the application's control, and hold suppliers and AI agents to the same standard.

- **Make recovery a rehearsed routine.**

Know where critical software runs, who repairs it and what keeps running while it is down. Measure how long rebuilding and restoring trustworthy data actually take.

Elias Mehri · RenX Management
Published 23 October 2026

[Read the article online / renx-management.com](https://renx-management.com)

References

1. W3Techs. Usage Statistics and Market Share of PHP for Websites, October 2026. 23 October 2026. [Source](#) [Return 1](#) [Return 2](#)
2. Calif. php2shell. [Source](#) [Return 1](#) [Return 2](#)
3. Wiz Research. PHP2Shell: AI uncovers critical RCE vulnerabilities affecting 70% of the Internet. 21 October 2026. [Source](#) [Return 1](#) [Return 2](#) [Return 3](#) [Return 4](#)
4. AWS Security Blog. Amazon Threat Intelligence identifies PHP2Shell exploitation prior to public disclosure in MadPot. 21 October 2026. [Source](#) [Return 1](#) [Return 2](#)
5. The PHP Foundation. Today's PHP Security Release, and How We're Strengthening PHP Security. 21 October 2026. [Source](#) [Return 1](#) [Return 2](#) [Return 3](#) [Return 4](#) [Return 5](#)
6. WordPress.org. Requirements. [Source](#) [Return 1](#)
7. W3Techs. Usage Statistics and Market Share of WordPress, October 2026. 23 October 2026. [Source](#) [Return 1](#)
8. Fortune. AI helped discover the PHP vulnerability that could take down 70% of the internet. Now, the race is on to patch it before AI-powered hackers attack. 21 October 2026. [Source](#) [Return 1](#)
9. UK National Cyber Security Centre. Critical security issue affecting PHP. 21 October 2026. [Source](#) [Return 1](#)
10. Al Jazeera, with Reuters. World leaders warn of security flaws found in PHP programming language. 22 October 2026. [Source](#) [Return 1](#)
11. NOS. Websites van overheid en in onderwijs offline vanwege lek, risico op hacks. [Source](#) [Return 1](#)
12. Anthropic. Disrupting an AI-orchestrated cyber espionage campaign. 13 November 2025. [Source](#) [Return 1](#)
13. Google Threat Intelligence Group. GTIG AI Threat Tracker: Advances in Threat Actor Usage of AI Tools. 5 November 2025. [Source](#) [Return 1](#)
14. VulnCheck. VulnCheck State of Exploitation 1H-2026. 28 July 2026. [Source](#) [Return 1](#)

15. Google Cloud / Mandiant. M-Trends 2026: Data, Insights, and Strategies From the Frontlines. 23 March 2026. [Source](#) [Return 1](#) [Return 2](#) [Return 3](#)
16. Palo Alto Networks. CVE-2026-109618 Impact of Critical Vulnerabilities in PHP (CVE-2026-109618, CVE-2026-109619) on Palo Alto Networks Products. 21 October 2026. [Source](#) [Return 1](#) [Return 2](#)
17. Palo Alto Networks Unit 42. Threat Brief: Salesloft Drift Integration Used To Compromise Salesforce Instances. [Source](#) [Return 1](#)
18. Wiz. Sha1-Hulud 2.0 Supply Chain Attack: 25K+ Repos Exposed. 24 November 2025. [Source](#) [Return 1](#)
19. OWASP Gen AI Security Project. OWASP Top 10 for LLM Applications 2025. November 2024. [Source](#) [Return 1](#)
20. php/php-src GitHub security advisory. RCE via OOM corruption (GHSA-w8hw-w9fx-j79w). 21 October 2026. [Source](#) [Return 1](#)
21. php/php-src GitHub security advisory. RCE via stale global state (GHSA-6wmf-4rv5-p3rp). 21 October 2026. [Source](#) [Return 1](#)
22. Kubernetes. Security Checklist. [Source](#) [Return 1](#)
23. OWASP Cheat Sheet Series. Authorization Cheat Sheet. [Source](#) [Return 1](#)
24. Kubernetes. Service Accounts. [Source](#) [Return 1](#)
25. OWASP Gen AI Security Project. LLM06:2025 Excessive Agency. [Source](#) [Return 1](#)
26. Google. Rust in Android: move fast and fix things. 13 November 2025. [Source](#) [Return 1](#)