Policy and engineering paper | For government officials

Reengineering National Cyber Resilience Around Existing Systems

A Two-Stage DevSecOps Policy Informed by PHP2Shell, the History of ARPANET, UNIX and Mach, and Memory Safety

Hideto Tomabechi

CyLab Carnegie Mellon University, C5I Center George Mason University, Cognitive Research Laboratories

26 October 2026

Contents

Abstract

Key messages for decision-makers

How to read this paper

1 Introduction: the problem and the thesis

2 What happened in October 2026: the facts of PHP2Shell

3 Why old assumptions remain dangerous: memory safety and C

4 What history shows: ARPANET, UNIX, BSD and Mach

5 Protecting PHP while continuing to use it: design conditions and emergency response

6 Server configurations and shared responsibility: the essence of the risk is the same on a VPS

7 Language options: what Common Lisp, Erlang and Rust guarantee, and what they do not

8 Alternative implementation in Common Lisp and historical experience

9 Isolation, restarts and serverless in practice

10 Making DevSecOps an operating model and an architecture

11 Measuring the effect: separating security from continuity

12 A mandate model and implementation sequence for governments

13 Using AI to carry out replacement safely

14 Conclusion: institutionalise the ability to change safely, without waiting for wholesale replacement

Appendix A. Reading the defensive code (all examples executed)

Appendix B. Deployment and operations templates for protecting existing PHP

Appendix C. Questions and answers for policymakers

Appendix D. Glossary

References

Abstract

In October 2026, two defects discovered at the core of the PHP runtime (CVE-2026-109618 and CVE-2026-109619, known as PHP2Shell) forced the simultaneous shutdown of web services in government, finance, transport, broadcasting and commerce across many countries, Japan among them. In Japan alone, at least 246 organisations, including national ministries, local governments, banks, an airline, electric utilities, universities and hosting providers, announced service suspensions. Government websites in Europe and Canada were taken offline as a precaution. An intrusion at a European government entity was also reported, but was withdrawn on 25 October because initial access through PHP2Shell could not be confirmed. Within the public information this paper has examined, no compromise with PHP2Shell as the initial access vector has been confirmed; the main harm has been service downtime. [52,53,57,58]

This paper treats the incident not as an isolated bug but as the expression of a structural problem. PHP applications are written in PHP, but the engine that runs them is written in C. C and UNIX were created for a small, mutually trusting research community, and they still delegate the correctness of memory allocation and release to the programmer. Public administration now runs on this foundation at scale, in an environment where anonymous attackers, including state actors, are permanently connected.

The world's operating systems and language runtimes cannot be replaced at once. The central thesis of this paper is that **without fundamentally changing the operating system or programming language, governments can redesign development, procurement, deployment and operations as DevSecOps, and in doing so shrink the attack surface, the probability of compromise, the reach of an intruder, the time to remediate and the time to recover; and these effects multiply.** As a second stage, high-risk components that process external input should be migrated progressively, with AI assistance, to languages and runtimes with strong memory safety and fault isolation, such as ANSI Common Lisp, Erlang and Rust.

The paper establishes the facts of the incident from primary sources and from inspection of the fix itself, explains the origin of today's risk through the history of ARPANET, UNIX, BSD and Mach, and proposes the controls and implementation sequence that governments should mandate. It also sets out impact and shared responsibility by server configuration and, drawing on an intrusion at a VPS service in Japan in the same week (whose route has not been disclosed), shows that the hosting arrangement, whether VPS or cloud, does not change this conclusion. It aims at a level of rigour that engineers can verify while giving policymakers plain-language explanations in every section. All code shown was compiled and run in the author's environment, and its output is reproduced.

Keywords: cyber resilience, DevSecOps, memory safety, PHP2Shell, zero trust, software supply chain, Common Lisp, Erlang, Rust, government procurement

Key messages for decision-makers

1. **The main damage was downtime.** Many organisations could not tell at once which of their servers ran PHP, in which version and configuration, and had no way to deploy a fix quickly and reliably. They were left with no option but to switch services off. This is a management and procurement failure as much as a technical one.
2. **The root cause lies in old design assumptions.** The PHP engine is written in C, where a memory-management error can lead directly to remote code execution. C and UNIX were designed around 1970 for a limited research community. The US and allied governments have already called for planned migration away from memory-unsafe languages. [39]
3. **There is no need to wait for wholesale replacement.** An asset register, automated deployment of fixes, least privilege, network segmentation, intrusion detection and recovery drills can sharply reduce the probability and scale of harm on the existing operating systems and languages. It is the same idea as keeping an old building in service with fire compartments, sprinklers, inspections and evacuation drills. Moving to a VPS or the cloud does not change this need (Section 6).
4. **"Substantial improvement" must be shown by measurement.** This paper does not claim an effect size. It proposes mandatory metrics, such as coverage of the asset register, time to deploy fixes, success in blocking privilege escalation and recovery time, measured before and after.
5. **Stage two replaces the riskiest components first.** Parsers of external input, privileged components and native extensions should move first to languages such as Common Lisp, Erlang or Rust. AI translation is a powerful aid, but only with an accountable human owner and independent testing and verification.
6. **Time has become the decisive variable.** Exploitation attempts were observed before disclosure, and millions of attempts were recorded within hours of the fix. AI was used in part of the discovery, and the same capability is available to attackers. A remediation cycle measured in weeks is no longer adequate.

How to read this paper

The paper proceeds from facts (Section 2), to causes (Sections 3–4), to principles of defence (Sections 5–9), to institutionalisation (Sections 10–13). Boxes marked For policymakers explain the essentials without jargon; boxes marked Technical note add detail for specialists. Appendix A explains the executed code line by line,

Appendix B provides configuration templates for existing PHP deployments, Appendix C is a question-and-answer guide for policymakers, and Appendix D is a glossary.

Factual statements carry reference numbers. Information on the October 2026 incident is changing daily; this paper reflects sources available up to 26 October 2026 (Japan time). Always recheck the latest official information before procurement or deployment decisions.

1 Introduction: the problem and the thesis

1.1 The gap between design assumptions and today's threats

On 1 January 1983, ARPANET switched its protocols from NCP to TCP/IP, a major milestone on the way to today's Internet. Its nodes were then limited to selected research institutions in the United States and a few sites abroad, such as NORSAR in Norway and University College London, connected in 1973. [10,63] UNIX (from 1969) and the C language (developed 1969–73, chiefly in 1972) were likewise built by Bell Labs researchers for their own research environment. [12,67]

Security awareness was not absent. RFC 602 of 1973 already warned of weak passwords, unauthorised remote use and access far wider than intended, problems that remain familiar today. [11] Yet designing languages, operating systems and protocols comprehensively around anonymous adversaries and foreign state actors permanently connected to the network was not the objective. C in particular, for the sake of performance and portability, made the correctness of memory allocation, release and bounds the programmer's responsibility. That design survives to this day for compatibility.

PHP2Shell in October 2026 is a case in which this gap between design-time assumptions and present-day threats surfaced as society-wide service disruption. PHP is a language of the Web era, evolved from tools published in 1995; it does not itself predate ARPANET. [21] But the engine that runs PHP is written in C and inherits C's properties.

1.2 Thesis and structure

The paper argues three points. First, most of the societal harm from PHP2Shell was amplified not by the defects themselves but by deficits in asset visibility, remediation, privilege separation and recovery. Second, these deficits can be reduced substantially through DevSecOps while keeping existing operating systems and languages. Third, in parallel, high-risk components that handle external input should be migrated preferentially to memory-safe languages and runtimes.

This is a policy and engineering paper grounded in primary sources and the author's research and development experience. Historical, specification and incident facts are referenced; engineering inferences and policy proposals state their conditions of application. The content of the fix was verified directly against the commit published in PHP's official source repository. [50]

1.3 Distinguishing kinds of evidence

Coverage of cyber incidents often conflates facts of different kinds. This paper distinguishes four. Stating which of them has actually been established is also a precondition for government communications that avoid both complacency and overreaction.

Category	Meaning	PHP2Shell example (as of 26 October)
Vulnerability	An exploitable defect exists	Unpatched PHP 8.2–8.5 and many end-of-life branches [1,52]
Attempt	Attack traffic has been observed	Observed by Amazon on 17 October, before disclosure; millions after disclosure [5,52]
Compromise	An attack succeeded and unauthorised actions took place	Not confirmed in public information; the European government case was withdrawn because the initial access vector could not be confirmed (Unit 42, 25 October) [53]
Precautionary shutdown	Service stopped for safety, regardless of compromise	At least 246 organisations in Japan; government sites in Europe and Canada [52,57,58]

The absence of a reported breach is not proof of safety. Equally, a shutdown alone does not justify inferring a data leak. The European government case, reported by Unit 42 on 23 October and withdrawn on 25 October, also shows how hard attribution is, that is, confirming which vulnerability was used for initial access. Because the attack took place in the context of a vulnerable PHP application, it was first judged to be PHP2Shell exploitation, but its link to initial access could not be confirmed. Early reports should be used for policy decisions only after checking their confidence and whether they have been updated. [53]

1.4 Threat model

The scope covers internet-facing services, internal APIs, development and deployment infrastructure, and supplier connections. Adversaries include foreign state actors and criminal groups, but also compromised domestic devices, stolen legitimate accounts and maliciously altered dependencies. The country inferred from an IP address is only supporting context; trust cannot be decided by the nationality or location of the source. Paths by which an external attacker acquires legitimate internal authority must be assessed at design time. This is the basic idea of zero trust. [38]

2 What happened in October 2026: the facts of PHP2Shell

2.1 What PHP is and why the impact was so wide

For policymakers

PHP is a programming language that runs on web servers and assembles pages in response to user input. Many of the systems used by governments and businesses, including WordPress, Drupal, Joomla and Magento, run on PHP. [57] When a user fills in a form or uploads a file, the PHP engine receives and parses the data before handing it to the application. These defects lay in that receive-and-parse stage, an entrance that every request passes through regardless of how the application is written. Careful application development could not have prevented them.

According to W3Techs, 69.2% of websites whose server-side language is known use PHP (as of 25 October 2026). [75] Estimates of the attack surface vary with method. Palo Alto Networks' Cortex Xpanse counted more than 22.6 million exposed instances that are potentially vulnerable. [53] Modat observed 4,384,302 PHP services on 1,886,612 hosts in the ten days before disclosure; about 79% of those with a readable version ran an unsupported branch. [55] Bitsight likewise reported that the most commonly observed branch was end-of-life PHP 7.4, and that only about a third of instances with an identified version were on the 8.2–8.5 branches covered by the fix. [62] None of these counts confirms vulnerability host by host, but the high share of end-of-life versions means that many organisations cannot simply upgrade to an official fixed release.

2.2 Timeline

The table lists the main facts bearing on this paper's conclusions (items without a time give the date only).

Date and time	Event
August 2026	Several researchers, including Calif, OpenAI and Wiz, independently report defects with the same root cause to PHP. AI-assisted analysis was reportedly used in part of the discovery. [52,55]
17 October	Amazon observes exploitation attempts in its MadPot honeypots before public disclosure. [5]
18 October	The PHP fix commit "Avoid state corruption via bailout" is authored (423 files changed on the 8.5 branch). [50]
20–21 October	Details leak before disclosure; information with an incorrect CVSS score and exploit code circulate. Cloudflare activates WAF (web application firewall; see the explanation below the table) rules at 03:02 UTC on 21 October. [52,60]

Date and time	Event
21 October, 21:55 JST	Prime Minister Takaichi, on X, urges preparation to patch that night and temporary suspension of functions where prompt patching is difficult. [56,61]
21 October	Government sites in Germany, Finland, Belgium, the Netherlands (the public health institute RIVM), France (prefectures) and Canada are taken offline as a precaution. CERT-EU advises temporarily disabling applications that cannot be updated. [8,57,58]
21 October, evening (JST)	PHP releases fixed versions 8.2.35, 8.3.36, 8.4.27 and 8.5.12, with minimal source backports for end-of-life 7.4, 8.0 and 8.1. [1,54,62]
21–22 October	Japan's Ministry of Finance suspends parts of its website and resumes on 22 October; the National Tax Agency and the Ministry of Health, Labour and Welfare also suspend systems. IPA and JPCERT/CC issue alerts. [4,6,7,52]
22 October	Cloudflare's CEO reports thousands of attempts in the ten hours before disclosure, tens of thousands within 45 minutes after, hundreds of thousands within two hours and millions by the time of posting. [52]
23 October	PHP advises disabling session upload progress pending a fix for a separate path not addressed by the 21 October releases (CVE-2026-115560). [1,54]
23 October	Palo Alto Networks Unit 42 reports that at a European government entity, password hashes and multi-factor authentication secrets were obtained within two hours of initial exploitation, followed by an NTLM relay into Active Directory and an attempt to gain domain administrator rights. It stated that the attacker's commands bore signs of an AI agent (withdrawn on 25 October). [53]
25 October	In an update, Unit 42 withdraws the 23 October addition. What it had observed was an attack on a European government agency carried out in the context of a vulnerable PHP application, which led it to believe that PHP2Shell had been exploited; it states that it cannot confirm that initial access was associated with the PHP2Shell CVEs. [53]

For policymakers: what a WAF (web application firewall) is

A WAF (web application firewall) sits in front of a web server, inspects the content of incoming requests (HTTP traffic) and blocks those that look like attacks. Whereas an ordinary firewall decides what to let through by addressing, "from where, to which door", a WAF judges by the content of a request: the shape and size of the data sent, or patterns characteristic of known attacks. It is often provided as a service by cloud providers such as Cloudflare, in front of their customers' sites.

The advantage of a WAF is that defences can be strengthened quickly by adding inspection rules, without changing the software itself. In this incident, Cloudflare was blocking attacks with WAF rules about nine hours before the fixed releases were published. [60] It also has limits. First, it has no effect on traffic that reaches the server without passing through it (internal networks, other entrances, direct connections to the server's address). Second, because its rules are based on the shapes of known attacks, attackers can slip past by changing the form of their requests; mutated attacks were observed in this incident too. Third, the defect itself remains on the server. A WAF is therefore one of the layers that buy time until the fix, not a substitute for it.

2.3 How the two defects work

For policymakers: the hotel key analogy

For efficiency, a PHP web server reuses work processes called "workers"; one worker handles many users' requests in turn, much as a hotel rents the same room to one guest after another. The working memory for each request is like the room a guest uses only during the stay.

One defect left the previous guest's room number in the hotel's register after check-out (the end of the request). The room had been cleaned and given to the next guest, yet someone relying on the old register could still open its door. An attacker could create this state deliberately with one request, then cause the "old key" to be used during a later request and execute arbitrary commands on the server. The other defect arose when processing was stopped abruptly for lack of memory: the room was abandoned half-prepared through an emergency exit, and the clean-up crew then treated the half-prepared room as if it were in order.

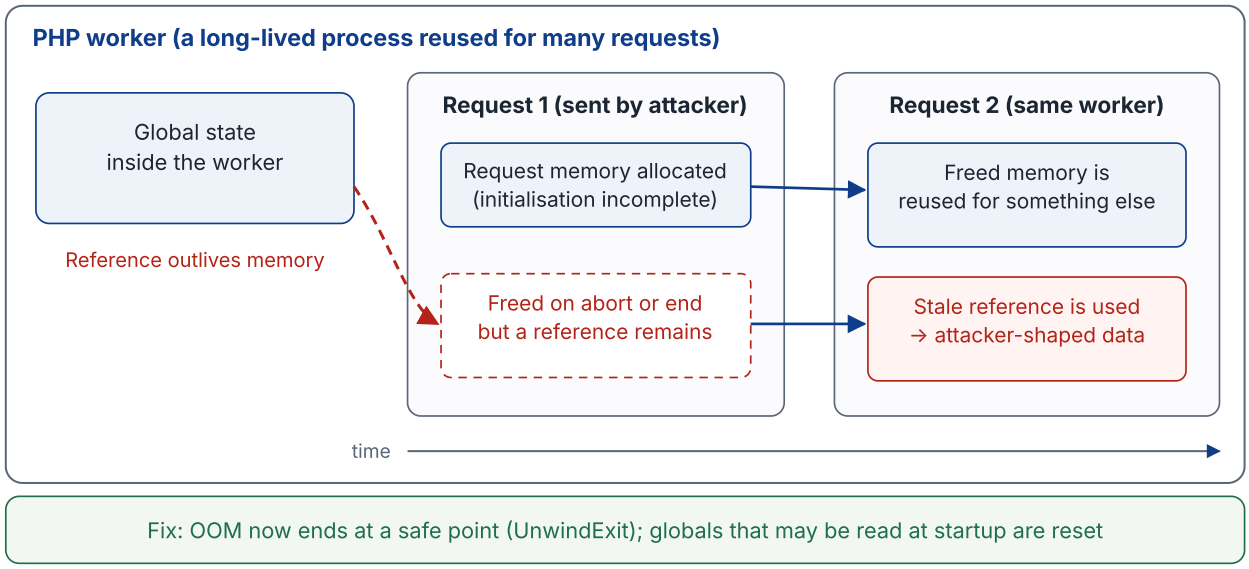

Figure 1. Worker reuse and stale references: a simplified view of CVE-2026-109619

Technically, the two defects can be summarised as follows. Both are memory-safety defects of the use-after-free (UAF) kind. [2,3,54]

Technical note: what a use-after-free (UAF) is

A C program allocates the memory it needs for its work and frees it, returning it, when it is finished. A use-after-free (UAF) is the error of reading or writing that memory again through an old address that survives the release (a dangling pointer). In C, freeing memory does not automatically invalidate the pointer, and any later use is undefined behaviour under the language standard.

The danger is that the memory manager may reallocate the freed region for another purpose. If an attacker can fill that region with data of their own through later input, the program treats that data, through the stale reference, as legitimate internal state. If the data includes values that tell the program where to go next, such as function pointers or virtual function tables, control flow can be hijacked and arbitrary code executed. Not every UAF leads to code execution, but this is the classic path from a UAF to remote takeover. In the hotel analogy, freeing is "check-out", the dangling pointer is "the old room number left in the register", and reallocation is "renting the room to the next guest". In CVE-2026-109619, a reference to memory freed at the end of a request remains in the worker's global state and is used by a later request handled by the same worker. Section 3 gives a minimal code example (Code 1).

Defect	Boundary failure	Qualification
CVE-2026-109618	An out-of-memory (OOM) interruption leaves partly initialised request state that shutdown processing then handles	With mod_php (embedded in Apache), a path before any application code runs was demonstrated. PHP-FPM behaves differently, for example terminating a child on startup failure; the same path must not be assumed.

Defect	Boundary failure	Qualification
CVE-2026-109619	The lifetime of request memory and the lifetime of global references held by the worker do not match	A later request in the same worker must consume the stale state. Not every path that produces stale state involves OOM.

If an attack succeeds, code runs with the web worker's privileges. That account varies between environments and is not necessarily www-data or root. But if that account can read data, database credentials or cloud credentials, the initial intrusion alone can be serious. And if the attacker uses that foothold to move laterally to the authentication infrastructure or backups, the damage can extend to the whole organisation.

2.4 The fix: from an emergency exit to orderly termination at a safe point

When PHP's engine (Zend Engine) hit a fatal condition such as exceeding the memory limit, it used C's `longjmp` mechanism to transfer control from the middle of processing straight to the top of the request (internally called `zend_bailout()`). `longjmp` does not run the clean-up that the skipped functions would have performed. [45] The fix commit, which the author inspected, describes itself as "Replace bailout on OOM with UnwindExit" and "Properly reset various globals that may be accessed during startup". [50]

Memory management after the fix has two tiers. When the ordinary (soft) limit is exceeded, the engine no longer escapes immediately: it records a pending out-of-memory condition, continues, and enters the regular termination path (UnwindExit) at a safe point in the virtual machine. If memory use reaches a higher (hard) limit in the meantime, the process writes a message to standard error and exits immediately with status 137 (`_exit(137)`). The hard limit is configurable through a new setting, `memory_hard_limit_percent`; by default it is 200% of memory_limit when that limit is at most 1 GiB, and memory_limit plus 1 GiB above that. [50] On the 8.5 branch the change touched 423 files and added about 11,500 lines, including regression tests for many distinct out-of-memory paths.

> **Technical note**
>
> `_exit(137)` terminates the process with exit status 137; it is not the same as sending SIGKILL (137 merely coincides with the value shells report for a SIGKILL termination). Nor does the fix remove every `longjmp`-style mechanism from PHP: it addresses the interruption path at issue, and the patched code still contains `zend_try` / `zend_catch` guards. [50]

Linux distributions often backport fixes without changing the version number. In the author's environment (Ubuntu 24.04 LTS), PHP reported version 8.3.6, which looks older than the fixed 8.3.36, yet the changelog of package 8.3.6-0ubuntu0.24.04.13 recorded the patch for these CVEs, dated 19 October. [51] Judging vulnerability by version number alone can lead to shutting down systems that are already fixed, or overlooking ones that are not. The asset register must record package-level fix history.

2.5 The 23 October update

On 23 October, PHP announced that a further remote code execution path (CVE-2026-115560), not addressed by the 21 October releases, exists for an uncommon combination of configuration and application behaviour in which the session serialisation handler is changed at runtime. Pending a fix, it advised disabling upload progress tracking with `session.upload_progress.enabled=0`; the Canadian Centre for Cyber Security issued the same mitigation that day. [1,54] The feature is enabled by default, and was enabled in the environment described above. Applying the 21 October updates therefore cannot be described as permanently securing every configuration and path.

2.6 The real picture of harm: shutdowns, attempts and compromises

Japan. A tally by a security researcher found that, by late October, at least 246 organisations had announced service suspensions, most of them precautionary. They include national bodies (the Ministry of Health, Labour and Welfare, the Ministry of Finance and the National Tax Agency), 18 local governments (including Sapporo, Yokohama, Mie Prefecture, Osaka Prefecture and Osaka City), 8 financial institutions, 11 transport and infrastructure operators (including an airline and electric utilities), 9 universities and 2 broadcasters, along with many game, video, e-book, e-commerce and hosting companies. [52] Major hosting providers patched every branch including old ones or disabled upload progress across the board, suspended control panels and other functions, and then restored them in stages. Individual outages lasted from hours to more than half a day. [52,61] As of 24 October, public information did not confirm any successful compromise in Japan attributable to these defects. [52] In the same week in Japan, unauthorised access involving the exfiltration of customer information was confirmed at a VPS service of GMO Internet; its intrusion route has not been disclosed, and no link to PHP2Shell has been confirmed (Section 6.6). [82,83]

Abroad. In Europe, Germany, Finland and Belgium shut down websites and services as a precaution, and sites of the Dutch RIVM and French prefectures were taken offline. [57] Some federal government sites in Canada were also temporarily taken down. [58] As for compromise, Unit 42 reported post-exploitation activity at a European government entity on 23 October, but withdrew it in a 25 October update, stating that initial access through PHP2Shell could not be confirmed. [53] Within the public information this paper has examined, no compromise with PHP2Shell as the initial access vector has been confirmed. One analysis also reports infrastructure assessed, with moderate confidence, as linked to North Korea probing unpatched hosts in twelve countries and regions, including Japan. [59]

> **For policymakers: downtime is harm too**
>
> Even when no data is stolen, stopping administrative procedures, financial transactions, health and care information services, airline bookings or e-book access imposes losses on citizens and the economy. Many organisations chose to shut down this time for three reasons: (1) they could not quickly establish where PHP ran and in which version, (2) they could not deploy a fix safely within hours, and (3) they could not be confident that an intrusion would not spread. Put the other way round, organisations that prepare these three capabilities in peacetime can protect themselves with minimal downtime. That is the starting point of this paper's proposals.

2.7 Five structural problems revealed by the incident

1. **A single point of failure in shared infrastructure.** Countless services worldwide depend on the same runtime, so one defect strikes across borders and sectors simultaneously.
2. **Lack of asset visibility.** Without knowing about PHP in supplier systems, packaged products, management tools and appliances, an organisation cannot even decide whether it needs to patch.
3. **Prevalence of end-of-life versions.** Much of the observed population runs unsupported branches that cannot simply be upgraded to an official fix. [55,62]
4. **Compressed time.** Pre-disclosure leaks and exploitation attempts, mass attack attempts within hours of disclosure, and AI-assisted discovery of the defects all coincided. The time allowed for deploying a fix has shrunk from days to hours. [5,52,55]
5. **Limits of perimeter defence.** A WAF provided a useful layer even before disclosure, but it does not cover traffic that bypasses it, and payload mutation was observed. A WAF is no substitute for the fix. [60]

3 Why old assumptions remain dangerous: memory safety and C

3.1 What memory safety means

> **For policymakers: once you return a room, stop using it**
>
> While running, a program "borrows" computer memory for its work and "returns" it when finished. Memory safety is the property that the language or runtime guarantees two things: a program never touches anything outside the range it borrowed, and never uses a place after returning it. C gives no such guarantee. A single mistaken line lets a program read or write a place that has already been reused for another purpose. Attackers exploit such mistakes to plant their own instructions there and make the program execute them.
>
> Since 2022, the NSA, the White House Office of the National Cyber Director (ONCD) and CISA have successively urged migration to memory-safe languages. [68,69,70] In a joint document of December 2023, CISA, the NSA and the FBI, with the authorities of Australia, Canada, the United Kingdom and New Zealand, cited reports that about 70% of Microsoft's vulnerabilities and about 70% of Google Chromium's were memory-safety issues, and called on software manufacturers to publish memory-safety roadmaps. [39] CISA and the FBI have stated that manufacturers should publish such roadmaps by the end of 2025. [71]

The two programs below show how C and Rust treat the same mistake (details in Appendix A.1). The C program compiles as it stands and, when run, reads memory that has already been freed. This is undefined behaviour under the language standard, which means that nothing that happens next is a violation of the standard. The error is reported at run time only when the program is compiled with the AddressSanitizer instrumentation. In Rust, the same mistake is rejected at compile time, and no executable is produced.

Code 1. C: a minimal use-after-free (UAF) `uaf.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    char *session = malloc(16);   // (1) borrow 16 bytes
    if (session == NULL) return 1;
    strcpy(session, "user=alice");
    free(session);                // (2) give the memory back (free)
    printf("%s\n", session);      // (3) use it after returning it: UAF
    return 0;                     //     undefined behaviour in C
}
```

Output (gcc 13.3, -fsanitize=address; addresses elided)

```
==25784==ERROR: AddressSanitizer: heap-use-after-free on address 0x… at pc 0x… bp 0x… sp 0x…
READ of size 2 at 0x… thread T0
freed by thread T0 here:
previously allocated by thread T0 here:
```

Code 2. Rust: the same mistake is rejected at compile time `moved.rs`

```rust
fn main() {
    let session = String::from("user=alice"); // (1) the variable owns the memory
    drop(session);                            // (2) ownership is given up; freed
    println!("{}", session);                  // (3) use after free will not compile
}
```

Compiler output (rustc 1.97.0)

```
error[E0382]: borrow of moved value: `session`
 --> moved.rs:4:20
  |
2 |     let session = String::from("user=alice"); // (1) the variable owns the memory
  |         ------- move occurs because `session` has type `String`, which does not implement
the `Copy` trait
3 |     drop(session);                            // (2) ownership is given up; freed
  |          ------- value moved here
4 |     println!("{}", session);                  // (3) use after free will not compile
  |                    ^^^^^^^ value borrowed here after move
  |
help: consider cloning the value if the performance cost is acceptable
```

The important point is that, even in C, analysis tools can find many such errors, but only if they are built into the development process and the tests actually exercise the faulty path. DevSecOps is the mechanism that turns such checks from a matter of individual diligence into a step that every change must pass (Section 10).

3.2 Assigning responsibility correctly

C is a language, GCC is a compiler collection, glibc is an implementation of the C standard library, and the Zend Engine is PHP's execution engine. Lumping these together as "GNU C" misplaces responsibility. CVE-2024-2961, for example, was a defect in glibc's iconv function that could write up to four bytes past the output buffer when converting to ISO-2022-CN-EXT; it is unrelated to the 2026 defects in PHP's request-state management. [9]

Memory safety, moreover, concerns mainly whether accesses inside a single process respect the bounds and lifetimes of objects. Protection through the operating system's virtual memory is a different layer that keeps one process from accessing another's memory; it does not prevent a UAF within the same process. PHP scripts do not normally handle raw pointers, yet defects in the engine or in native extensions can still be reached through external input.

3.3 longjmp and guarantees of clean-up

C's `setjmp` / `longjmp` transfers control non-locally to a saved point of execution. Using it is not always undefined behaviour in itself; it becomes undefined when there is no corresponding call to `setjmp`, when the function that contained it has already returned, or when `setjmp` was called in another thread. [45] The problem is that `longjmp` performs none of the clean-up of the functions it skips. GCC's own `__attribute__((cleanup))` likewise runs when a variable leaves its scope normally (and during exception unwinding), not when control escapes by `longjmp`. [77] The more a piece of code relies on clean-up, the more likely it is to leave a half-finished state behind after a forced escape.

There is no basis for the claim that GCC optimisation generally guarantees a memory layout that is easy to attack. The essence of this incident is not specific to the GNU environment. It is a structural problem common to long-lived processes written in C: a design in which a forced escape skips clean-up, combined with references that survive across requests.

The guarantees that different languages offer for clean-up on escape compare as follows. None of them guarantees clean-up across a power failure or a forced termination by the operating system (SIGKILL).

Mechanism	Clean-up on escape	Outside the guarantee
C: `longjmp`	None (intermediate functions are skipped)	All clean-up is the programmer's responsibility
C++: exceptions	Destructors run during unwinding	`std::terminate`, `abort`, forced termination
Common Lisp: `unwind-protect`	Clean-up runs on both normal exit and non-local exit [23]	Corruption of the implementation, forced termination by the OS, non-standard forcible termination of threads
Rust: `Drop`	Runs at end of scope and during panic unwinding	The `panic=abort` setting, forced termination of the process (measured in Appendix A.6)

Mechanism	Clean-up on escape	Outside the guarantee
Erlang: process exit	The process heap is reclaimed and monitors are notified	Memory exhaustion of the whole VM, failures in native functions (NIFs) [27]

4 What history shows: ARPANET, UNIX, BSD and Mach

For policymakers

Much of the foundation of today's information systems was built in the 1970s and 1980s, in environments where a limited number of researchers trusted one another. The engineers of the time were not incompetent; their design goals were different. The foundation has been improved for half a century while preserving compatibility, but its fundamental design philosophy cannot easily be changed. The lesson of history is not that "nobody thought about security back then" but that "there is a gap between design-time assumptions and present-day threats, and that gap must be closed through operations and structure".

4.1 From ARPANET to the Internet

RFC 801 set the goal of a complete transition from NCP to IP/TCP by 1 January 1983. [10] The date is an important milestone, but research on and operation of interconnected networks had been under way before it, and the account that "ARPANET suddenly turned into the Internet on that day" is too simple. On 15 June 1973, NORSAR in Norway became the first ARPANET node outside the United States, connected via a satellite link, and University College London (UCL) in the United Kingdom followed shortly afterwards. [63] Participation in the early network was limited, but international connections and security concerns both existed. RFC 602 warned of easily guessed passwords, defencelessness against unauthorised remote use, and access from a far wider range of places than intended. [11]

4.2 UNIX and C

According to Dennis Ritchie, UNIX was born in 1969. [12] C took shape between 1969 and 1973, and its most creative period was 1972. [67] UNIX did provide access control for many people sharing one computer, but that differs from comprehensive defence in today's environment of permanent connection to an unknown multitude of hostile parties. C gave priority to near-hardware efficiency and portability, and left the correctness of memory bounds and lifetimes to the programmer. That choice still underpins most operating systems, language implementations and network devices half a century later.

Aspect	At design time (1970s)	Today
Users	Researchers and engineers who knew one another	An anonymous multitude; automated attack infrastructure

Aspect	At design time (1970s)	Today
Assumed adversary	Internal misuse, pranks, leaked passwords	State actors, criminal organisations, AI-assisted automated attacks
Connectivity	Leased lines, a limited number of sites	Always-on connection, cloud, contractors, IoT
Distribution of fixes	Tapes and individual contact	Immediate deployment while attacks begin within hours of disclosure
Memory management	Left to the programmer for performance	The same design, inherited for compatibility, is a leading cause of serious vulnerabilities

4.3 BSD and the spread of networking

BSD developed in the late 1970s as add-on software for UNIX, and 4.2BSD, released in August 1983, contributed greatly to the spread of TCP/IP and the sockets API. [14,79] This is not the same as inventing TCP/IP or producing every first implementation. BSD adopted TCP/IP to connect to research networks; security design for a hostile environment was not its main focus. vi and csh were early BSD products, whereas OpenSSH was released by OpenBSD developers in December 1999, a different era. [16] The licences required for early AT&T-derived code should also be distinguished from the licences of the later redistributable BSD code. [15]

4.4 Commercial UNIX and Linux

It is inaccurate to define AT&T UNIX as a whole as SVR4 (System V Release 4). SVR4 was a late release, shipped in 1989, that merged System V, BSD and Xenix. Nor was the 1993 sale of USL (UNIX System Laboratories) to Novell the first time other companies were allowed to use UNIX: commercial UNIX had been licensed to many vendors long before, and Sun was both a competitor and a partner in the merger. [13] In December 2000, IBM announced that it would invest about one billion dollars in Linux the following year, a major turning point for enterprise use of Linux. [17] Market adoption, however, is no proof of security. Many of today's public administration systems run on Linux, using many components written in C.

4.5 The lesson of 1988: the Morris worm

In November 1988, the Morris worm spread across UNIX-family computers on the Internet. According to Spafford's analysis, the worm broke in by combining a buffer overflow in fingerd's input handling (the use of `gets`, which does not check bounds), sendmail's debug feature, and trust relationships between computers together with password guessing. [64] That the lack of memory safety in C can translate directly into large-scale harm across a network has been known for 38 years. The response since then has been to accumulate fixes for individual defects, mitigation techniques and operational defences. PHP2Shell belongs to the same lineage.

4.6 Mach: rebuilding internal boundaries while keeping compatibility

Carnegie Mellon University's (CMU) Mach project began in 1985 and ran until 1994. [18] The kernels of the 4.2BSD-based UNIX systems of the time had become hard to understand as features not envisaged at the outset were added. Mach aimed at interprocess communication (IPC), virtual memory, lightweight threads, support for many processors, portability and compatibility with UNIX. It was first implemented by incorporating BSD code, and later moved to a structure in which UNIX functionality ran as applications outside the kernel. CMU researchers wrote that they had been running this separated environment on campus since March 1989. [19] The Mach lineage spread through NeXTSTEP, OSF/1 and other systems.

IPC through ports (communication endpoints) makes it possible to design explicitly "which party is given authority to request what". A smaller kernel is an opportunity to narrow the scope of privileged code, but mistakes in each server's privileges, message validation, drivers or handling of shared resources make it vulnerable. Having "divided" a system is not the same as having "demonstrated a secure boundary".

Apple's XNU is a hybrid kernel that combines CMU's Mach kernel with components derived from FreeBSD and a C++ API for writing drivers (IOKit). [20] That the Mach lineage survives in macOS and iOS does not mean that a pure microkernel structure is used as such. Mach-O, incidentally, is an executable file format, not the kernel itself.

> **Policy lesson:** Mach showed that internal boundaries of responsibility can be rebuilt while preserving externally visible compatibility (the UNIX API). In public administration systems too, the boundaries of privilege, communication and failure can be redesigned while existing applications keep running, without waiting for wholesale replacement. This is the idea behind the first, DevSecOps stage.

5 Protecting PHP while continuing to use it: design conditions and emergency response

PHP developed from "Personal Home Page Tools", released by Rasmus Lerdorf in June 1995. [21] The history of PHP itself should be assessed separately from the risks of today's engine, extensions and applications. Even for the same PHP application, the losses from a compromise vary greatly depending on whether the runtime is patched, unnecessary extensions are removed, privileges are minimised, database access is restricted and secrets (passwords and keys) are short-lived.

5.1 Order of emergency response

1. **Identify:** Establish public-facing assets, how PHP runs (mod_php, PHP-FPM, CGI and so on), its extensions, and the fix status of each package (its fix history, not its version number). Include PHP embedded in contractors' systems, packaged products, management tools and appliances.
2. **Fix:** Apply the official fixed release or the distribution's backport. Updating a CMS or plug-ins is no substitute for updating the PHP runtime. End-of-life versions require a fix from the supplier or migration to a

supported branch.

3. **Restart:** Restart every worker, so that no process holding stale state remains.
4. **Verify:** Check the configuration actually running (version, settings and `session.upload_progress.enabled`). The CLI and web server configurations may differ.
5. **Investigate:** Preserve logs from before and after disclosure, and look for suspicious processes, files, outbound connections and accounts. A fix does not undo an intrusion that has already succeeded. [4,54]

5.2 Limits of interim measures

`memory_limit=-1` (no limit) is an interim measure that the official advisory described for specific conditions; it is not a safe setting for everyone. Removing the limit leaves other paths open, such as service outages through memory exhaustion and failures of basic allocations, so applying the fixed release takes priority. [2] A limit on input size does not necessarily bound how much data structures grow during processing either. nginx's `client_max_body_size` and `client_body_buffer_size` govern the acceptance of request bodies; they are not limits on PHP's internal memory use. [47] A WAF was a useful additional layer of defence even before disclosure, but it has no effect on paths that bypass it. [60] Interim measures buy time; they are no substitute for the fix.

> **For policymakers: what organisations that stayed online had in common**
>
> An organisation that can, within hours of a fixed release, distribute a tested update to every server, replace every worker and confirm the running configuration has no need for a long shutdown. What makes this possible is not individual skill but an asset register, an automated build, test and deployment pipeline, procedures for staged switchover, and regular peacetime drills. Section 10 describes these mechanisms concretely.

6 Server configurations and shared responsibility: the essence of the risk is the same on a VPS

The impact of PHP2Shell is not determined by the type of operating system or the name of the hosting arrangement. Whether on Linux, Windows Server, a VPS, a dedicated server, shared hosting or a container, any unpatched PHP that processes web requests can be in scope. PHP's official notices, too, treat the various modes of handling web requests broadly as subject to update. [1] At the same time, the same attack has not been demonstrated on every operating system and execution mode. As with the distinctions between kinds of evidence in Section 1.3, "in scope for updating", "conditionally affected" and "demonstrated attack path" must be kept apart, and general guidance alone must not be used to certify that a particular environment is safe.

6.1 Judge along four axes

1. **Operating system:** Linux, Windows Server, the BSD family and others. The OS name alone does not determine the state of PHP.
2. **PHP execution mode (SAPI):** embedded in Apache (mod_php), PHP-FPM, FastCGI under IIS, CGI, FrankenPHP, LiteSpeed and so on.
3. **Deployment form:** physical server, VPS or virtual machine, container, shared hosting, PaaS, serverless.
4. **Operational responsibility:** who updates PHP and carries the work through to replacing every worker and verifying the running state.

These categories are not mutually exclusive. A combination such as "IIS FastCGI on a Windows Server VPS, maintained in-house" is entirely possible. Whether action is needed must never be decided on one axis alone, for example "because it is a VPS" or "because it is Windows".

> **For policymakers: a VPS is "a flat in a rented apartment block"**
>
> A VPS (virtual private server) is a hosting arrangement in which one large physical server is partitioned by virtualisation and rented to each customer as if it were a dedicated server. In terms of an apartment block, virtualisation corresponds to the walls between flats. Thick walls make it harder for a fire or a burglar next door to reach your flat.
>
> But walls do not make up for leaving your own front door unlocked. If the operating system or PHP running inside a VPS has a defect, an attacker has no need to cross any wall and simply comes in through the front door. Moreover, an apartment block has a management office, a master-key cabinet and shared facilities that reach every flat. For a VPS, these are the provider's control panel, the systems that create and restore servers, and the infrastructure for remote access and monitoring. If these are breached, many flats are put at risk at once, however thick the walls. Moving to a VPS is therefore no reason to consider a system safe. The DevSecOps mechanisms described in Section 10 are needed both inside each flat (updates and privilege management by the customer) and in the management office (the provider's management infrastructure).

6.2 Configuration-by-configuration assessment

The following table is an operational arrangement that combines the official update scope with the specifications of each platform. It is not a list of attacks demonstrated against particular providers or products.

Configuration	How to think about the impact	Main checks and actions
Linux + Apache (mod_php)	Update first. An attack path before any application code runs has been demonstrated. [2]	The PHP module actually loaded, the distributor's fix, replacement of Apache workers

Configuration	How to think about the impact	Main checks and actions
Linux + nginx or Apache + PHP-FPM	FPM behaves differently in some respects, but cannot be declared safe. [2,3]	FPM version, pools, extensions, startup settings, replacement of all child processes
Windows Server + IIS (FastCGI)	Unpatched PHP is in scope. Do not generalise results demonstrated on Linux to every Windows configuration.	The php-cgi.exe the handler points to, the php.ini actually applied, application pools and execution identities [86]
Windows + Apache or bundled stacks	Products that bundle PHP are in scope too. Windows Update does not update separately installed PHP.	Connection mode, bundled PHP, extension DLLs, distribution package, old processes
Self-managed VPS or cloud VM	PHP runs inside the virtualisation boundary. Stealing data inside a VPS requires no escape from virtualisation.	Who updates the guest OS and PHP, execution privileges, reach to databases and backups [87]
Dedicated physical server	Avoiding co-tenancy does not remove defects in PHP itself.	Multiple services on the same machine, shared secrets, separation of administrative privileges
Shared hosting or managed VPS	The contractual scope of maintenance is decisive; the label “managed” alone settles nothing.	Evidence that the provider fixed PHP and updated all processes; customer-side CMS and plug-in updates
Docker or Kubernetes	PHP inside containers is in scope. A restart alone brings back the same vulnerable image.	Rebuild fixed images, update all replicas, confirm running digests (Section 9)
PaaS or serverless	Responsibility depends on whether the runtime is provider-managed or customer-supplied. Environments are not necessarily fully reinitialised for each request. [32]	The actual supplier of PHP, custom images or layers, deployed versions and reuse behaviour
BSD family and other systems	Do not assume immunity from the OS name. Exploitability depends on the build and configuration.	Fixes in packages, ports or custom builds. Jails share the host kernel, so update inside each jail separately [88]
Parts that do not use PHP	Static sites, ASP.NET and other parts that do not run PHP are not directly in scope.	Whether PHP remains in co-hosted admin screens, old sites or back ends

6.3 Linux servers: identify the PHP that is actually running

nginx and Apache are entrances for requests; their names alone do not determine how PHP runs. Distinguish nginx passing requests to PHP-FPM, PHP embedded in Apache (mod_php), and Apache passing requests to PHP-FPM. With mod_php the problem is reached during request parsing, so updating the CMS or relying on

login authentication offers no protection. [2] PHP-FPM differs from mod_php in that it terminates a child whose request startup fails, but other conditions, such as processing after a successful startup, apply as well, so this is no reason to skip the update. [2,3]

Checks should cover the distributor's fix information (fix history, not version numbers, as Section 2.4 explains) and also bring into the register PHP installed from source, from other repositories and in multiple copies under control panels. Then confirm that the update has reached the running processes. Code 16 in Appendix B.7 performs this check mechanically from the information in `/proc`. It is also essential to separate execution users and database privileges per site and to limit secrets and writable areas. Even without root being taken, whatever the web worker's privileges can read can be stolen.

> **Technical note: "updated" is not the same as "running the updated code"**
>
> A Linux package update normally deletes the old files and puts new ones in place. Processes that are already running keep executing the deleted old executable or libraries they have loaded, so the target of `/proc/<PID>/exe` or entries in `/proc/<PID>/maps` show "(deleted)". With mod_php, PHP itself is a shared library (libphp) loaded into Apache; with PHP-FPM, the master process and the children of each pool run PHP. In both cases the fix is not complete until the replacement of every process has been confirmed. Where compromise is suspected, however, decide before restarting whether evidence needed for investigation (memory, logs, suspicious files) must be preserved (Section 5.1).

6.4 Windows Server: separate the updates of IIS and PHP

PHP2Shell is a problem inside the PHP runtime (the Zend Engine), not a defect that depends only on Linux's glibc. [50] On Windows Server, too, check the fix status of the PHP in use. With FastCGI under IIS, the handler mapping specifies the php-cgi.exe that processes requests, and the FastCGI definition sets that process's arguments and environment variables. [86] Unpacking a new version into another folder does not switch anything over. Check the combination of PHP build (x64 or x86, thread-safe or not) and extension DLLs, the php.ini actually applied, and the FastCGI arguments and environment variables; after updating, confirm that no old php-cgi.exe processes remain. Separate application pools and restrict execution identities and NTFS permissions. Code 17 in Appendix B.8 identifies the actual PHP from the IIS configuration file and records its version and hash.

> **Technical note: the 26 October Windows rebuilds and the "version number trap"**
>
> On 26 October, PHP rebuilt its Windows packages as 8.2.35-1, 8.3.36-1, 8.4.27-1 and 8.5.12-1. This updated the bundled compression library Brotli to 1.2.1, fixing a flaw (CVE-2026-111497, an out-of-bounds write in decompression), and is separate from the PHP2Shell fixes. The flaw can be reached when the cURL extension has automatic decompression of HTTP responses enabled; this is off by default in cURL, but applications and HTTP client libraries may turn it on. [84,85] Because the PHP source and runtime versions are unchanged, `php -v` prints the same before and after the rebuild. That the displayed version cannot identify the contents of a package is the same structural problem as the Linux backports discussed in Section 2.4. This is why the source, build number and SHA-256 hash of distributed packages should be managed in the software bill of materials (SBOM) and the asset register.

6.5 Shared responsibility for VPS, dedicated servers and shared hosting

A VPS is a form of deployment and provision; it is neither an operating system nor a mechanism for fixing PHP. A VM-based VPS has a boundary per guest OS, but services using OS-level virtualisation may share the kernel. Whatever the isolation method, access control is separately needed between PHP and the data inside. If code is executed with web worker privileges, the configuration files, database credentials, API keys and upload areas readable with those privileges are within the scope of damage. A serious incident can occur even if root or Administrator privileges are not taken and no escape to the physical host succeeds. Conversely, a PHP compromise does not automatically mean complete control of the whole host.

Arrangement	Who updates	Evidence the customer should check
Self-managed VPS or IaaS	Normally the customer manages the guest OS and the PHP it installed; the actual contract prevails. [87]	Fix packages, time of application, the version running after restart
Managed VPS or dedicated server	Confirm by contract whether the provider's maintenance scope includes PHP, extensions and versions under the control panel.	List of targets, status of application, open exceptions, owners and deadlines
Shared hosting	Often the provider handles PHP and the customer the application; settle it by contract.	Fix of the selected PHP, worker updates, handling of old versions; a CMS update alone does not complete the work

Adding memory to a VPS is no fundamental fix. Conversely, making `memory_limit` unlimited on a small machine raises the risk of other outages (Section 5.2). Per-request limits, concurrency and host capacity should be designed separately.

6.6 What the GMO case in Japan shows

In the same week as PHP2Shell, the incident most widely reported in Japan was unauthorised access to "Onamae.com Desktop Cloud", a VPS service for automated foreign-exchange (FX) trading provided by GMO Internet. According to its announcements and press reports, events unfolded as follows.

Date	Event
19 October	Investigation begins after an inquiry from a customer. [83]
20 October	Confirmation that a third party had installed a malicious program in some customers' server environments; new sign-ups and the control panel are suspended. [81,83]
21 October	From 04:00, servers on all plans are stopped in turn. Servers potentially affected: 26,739 (not a count of confirmed damage). [81,83]
22 October	Announcement that login credentials for trading accounts and the like may have been viewed and misused. The Financial Services Agency notifies FX firms through the Financial Futures Association of Japan, asking them to alert customers and monitor trading. [82,83]
23 October	Customers are asked to terminate all sessions (because existing sessions can survive a password change) and, as a precaution, to back up and virus-scan their own devices. [82,83]
24 October	GMO announces that it has confirmed that customer information was taken out (type and number undisclosed). It states that it received a monetary demand from a person believed to be the attacker and will not comply. Compensation for losses from unauthorised trading and a payment to all customers are announced. The existing platform will not be restarted as it is; customers are offered, with their consent, migration to an alternative service on an independent cloud platform. GMO designates the case the "first activation" case of active cyber defence and asks the relevant government bodies to trace, block and neutralise the attack infrastructure. [82,83,89]
25–26 October	GMO collects customers' devices for analysis by its white-hat team and again asks customers to fully scan and back up their devices. It reports a threat to publish customer information and launch a ransomware attack at 12:00 on 27 October. [90]

The intrusion route has not been disclosed. GMO has stated that "a security problem existed in the system of this service", but has not said whether PHP was running or whether PHP2Shell was involved. [89,90] The malicious program was confirmed on 20 October, before the PHP2Shell fixes were published (the evening of 21 October) and after pre-disclosure exploitation attempts were observed (17 October), but timing alone cannot establish a link. This paper does not count the case as PHP2Shell damage. Claims made by the attacker side, such as the routes of intrusion and data theft, are also unverified.

Even so, the case matters for this paper's argument. First, even with a VPS as the hosting arrangement, the damage reached inside customers' servers. The malicious program was placed in customers' server environments, and trading-account credentials stored on the servers were targeted. Virtualisation boundaries separate customers from one another, but they do not protect the software running inside each server or the secrets kept there. The provider is also collecting and analysing customers' own devices, so concern now extends to those devices too. [90] Second, to prevent the damage from spreading, the provider stopped the servers on all plans

(26,739 servers were described as potentially affected). Control panels, server creation and restoration, and remote-access infrastructure that reach many servers form common management infrastructure, where a breach at one point can spread to the whole. Which part was breached in this case has not been disclosed, but the fact that a complete shutdown had to be chosen illustrates the danger. Third, the provider decided not to restart the existing platform as it was and offered customers migration to an alternative service on an independent cloud platform. [89] In not continuing to use a compromised environment and rebuilding from a trustworthy state, this is in line with the immutable infrastructure of Section 10.

So whether or not PHP was running, the questions the case put to organisations are the same as those of PHP2Shell. What runs where (the asset register)? Who fixed what and when, and did the fix reach the running systems (responsibility and evidence for updates)? Which secrets sit on servers, and how far can an intruder reach (least privilege and secret management)? Can credentials and sessions be revoked immediately (response)? Can environments be rebuilt from trustworthy artefacts (recovery)? Moving to a VPS only redistributes these questions between provider and customer; it does not remove them. A DevSecOps capability is essential on one's own servers, on a VPS and in the cloud alike.

Technical note: the three layers of a VPS and how to defend each

Think of VPS security in three layers. (1) Inside the guest: the operating system, runtimes such as PHP, applications and stored secrets. This is the customer's responsibility for updates and least privilege (or the provider's, depending on the contract, for managed services). (2) The isolation layer: the hypervisor or container runtime. It separates customers but does not prevent defects inside a guest; OS-level virtualisation also shares the kernel. (3) The management layer: control panels, server creation, cloning and restoration, relays for remote desktops and consoles, and monitoring and backup agents. Because these can access many guests with privileges, they need separation of duties, multi-factor authentication, short-lived credentials, and recording and monitoring of changes.

In incident response, changing passwords may not be enough. In the GMO case, too, customers were asked to terminate all sessions, because existing login sessions to trading accounts could survive a password change. [83] Response procedures must include revoking sessions and tokens, not only changing credentials.

For policymakers: "because it is a VPS" or "because it is a big provider" is no reason to feel safe

Because the GMO case and PHP2Shell occurred in the same week, they are easily confused, but within the facts that have been made public, no link between them has been confirmed. Policy discussion must first keep the two apart.

The lesson they share, however, is clear. Even when servers are rented as a VPS or in the cloud, someone must take responsibility for updating the software running inside, managing the secrets stored there, and containing and rebuilding after a compromise. When government bodies procure VPS or cloud services, they should require the following four points by contract, regardless of the provider's size or name, and check them regularly: (1) the scope of and deadlines for updates; (2) evidence that updates have reached the running environment; (3) protection of the management infrastructure (multi-factor authentication, separation of privileges, monitoring); (4) procedures and drill records for notification, session revocation and rebuilding after a compromise. Action against attack infrastructure under active cyber defence is an important new tool, but it does not replace the victim organisation's own capability to contain and recover.

6.7 Other configurations to watch

Configuration	Points to watch
Containers and Kubernetes	Ordinary Linux containers share the host kernel. The host OS is not the only thing to update: fix and redeploy the images containing PHP and check the running image of every replica. Automatic restarts and periodic replacement are not patches. Persistent volumes, databases and stolen credentials are not fixed by regeneration (Section 9).
PaaS and serverless	Register provider-managed runtimes separately from customer-supplied PHP binaries, containers and layers. Execution environments, global state and `/tmp` may be reused. [32] Shortening execution time does not remove the risk of secrets or database privileges readable during that request.
FrankenPHP, LiteSpeed, CGI and CLI	Check other implementations that embed PHP as subject to update; do not apply explanations for mod_php or FPM to every implementation as they stand. CGI, which ends with each request, and FastCGI, which handles many requests, differ in the conditions under which state survives between requests, but the failure of one path does not imply that PHP as a whole is safe. CLI-only batch jobs do not have the same exposure as public HTTP servers, but assess them if they take external input or run long-lived custom workers.
Behind CDNs, WAFs and load balancers	Blocking and size limits at the entrance are supplementary. Close direct paths to the origin server too. If there are multiple back ends, check every one. Where PHP parses requests before authentication, application login does not protect. [2]

6.8 Items to confirm with administrators and contractors

With the following records in hand, “we think we updated” can be distinguished from “the update is reflected in the running configuration”. They can also serve as items government bodies require contractors to submit.

1. **Exposure:** target domains, admin screens, APIs, direct paths to the origin, old sites
2. **OS and deployment:** operating system; physical, VPS or container; virtualisation method; who maintains it
3. **What actually runs:** PHP executable or module, SAPI, version, extensions, execution identity
4. **Evidence of fixes:** distributor advisories, package versions, backport records, image digests, package hashes
5. **Settings in effect:** actual startup settings and site-specific overrides, upload progress tracking, capacity and concurrency
6. **Processes in effect:** records that every worker, pool and replica was replaced
7. **Functional checks:** login, upload, APIs, sessions, payments and other key services
8. **Scope of damage:** secrets readable from that PHP, database privileges, writable areas, permissions to operate backups
9. **Investigation:** preservation of relevant logs and change history; examination of suspicious processes, files and connections
10. **Recovery:** isolation when compromise is suspected, revocation of credentials and sessions, rebuilding from trustworthy artefacts, verification of data integrity

Applying a fix closes a known entry route; it does not prove that no compromise happened in the past. Avoid restarts that merely destroy evidence, and balance urgent containment with preserving the evidence that is needed. What must be asked is not the name of the server configuration but whether patched PHP is actually running, whether privileges are limited in case of compromise, and whether updates and recovery can be repeated. This is the precondition for isolation in Section 9 and DevSecOps in Section 10.

7 Language options: what Common Lisp, Erlang and Rust guarantee, and what they do not

As destinations for the second stage, this paper treats ANSI Common Lisp, Erlang and Rust as strong candidates. Each, in its own way, structurally reduces a family of defects inherited from C. Adopting a particular language is not, however, treated as proof of security.

Option	Useful mechanisms	What is not guaranteed
ANSI Common Lisp	Garbage-collected objects; the condition system (conditions and restarts); clean-up on escape through `unwind-protect`; redefinition in a running system	Unlimited memory; clean-up on forced termination by the OS; freedom from defects in the FFI (foreign calls) or the implementation itself; automatic transactional consistency

Option	Useful mechanisms	What is not guaranteed
Erlang/OTP	Lightweight processes with separate heaps; message passing; localisation of and recovery from failures through monitoring and supervision; hot code replacement	Isolation from memory exhaustion of the whole VM or failures of native functions (NIFs); restart within a bounded time; automatic reversal of external side effects
Rust	Static guarantees, within safe code, of memory bounds and lifetimes through ownership, borrowing and types; clean-up through `Drop`	Automatic verification of `unsafe` code or FFI; elimination of memory leaks, deadlocks or authorisation errors; clean-up under `panic=abort`

Common Lisp. A `handler-bind` handler runs without unwinding the stack, in the dynamic context of the place where the error was signalled. The handler can examine the situation and invoke a "restart", a recovery option prepared by a lower-level function. When control transfers to a restart, the stack is unwound properly up to that point, and the clean-up forms of any intervening `unwind-protect` are run. [23,24,25] The language thus supports a division of labour in which the upper level decides *how* to recover and the lower level, where the problem occurred, defines *where* recovery can take place (Appendix A.4). What `unwind-protect` guarantees, however, covers normal exit and the non-local exits that the language handles; it does not extend to power failure, SIGKILL, corruption of the implementation or non-standard forcible termination of threads. Clean-up itself can also fail.

Erlang. Erlang processes are not operating-system processes but lightweight processes running inside the same virtual machine (BEAM). Each process has its own heap and cannot refer directly to the memory of another. The "let it crash" design, in which only the failing process is terminated and a supervisor starts a fresh one, achieves strong localisation of failures within pure Erlang code. [26] If a native function (NIF) crashes, however, the whole VM stops. [27] Restarts by supervisors must also be combined with limits on restart frequency, ordering of dependencies and designs that prevent duplicate processing.

Rust. Rust's safety guarantees assume that the inside of `unsafe` blocks, the compiler and libraries keep the necessary promises. [28] By default, a panic unwinds the stack and runs `Drop` for the values along the way, but no unwinding can be expected under the `panic=abort` setting, termination due to allocation failure, or forced termination by the operating system. [29] This difference was confirmed experimentally for this paper (Appendix A.6). The absence of garbage collection does not mean the absence of any run-time processing either.

It cannot be asserted that there is a single direct lineage from Common Lisp through Erlang and Rust to Kubernetes. This paper compares them as different solutions to common design problems: localising failure, owning resources and recovering.

8 Alternative implementation in Common Lisp and historical

experience

8.1 The option of implementing PHP on a managed runtime

Had the PHP runtime been built not on manual memory management as in C but on a strong managed runtime such as ANSI Common Lisp, the family of defects seen in this incident would very likely have been avoided at the implementation stage. In Common Lisp, memory is managed by garbage collection and programs do not handle raw pointers. The very state of a reference pointing to freed memory cannot be created within the safe language. Interruptions of processing are handled by the condition system and `unwind-protect`, and the language guarantees the clean-up along the way.

This direction also has historical support. The CMUCL (CMU Common Lisp) manual records that its development at Carnegie Mellon University was funded by DARPA (the US Defense Advanced Research Projects Agency) under the “Research on Parallel Computing” contract, and that it provided a native-code compiler, a garbage collector and an FFI. [46] This provenance is important context for the technology choice, but having been developed with military funding is not in itself proof of security.

The paper’s claim can be stated as a reasonable design hypothesis: if a language implementation equivalent to PHP is built in Common Lisp and restricted to safe object operations and explicit state transitions, there is room to avoid, at the implementation level, the family of defects that stem from manual deallocation and raw pointer manipulation. The ANSI standard, however, does not specify an unlimited heap or per-request isolation of resources. Embedding libraries written in C into the same process through the FFI also imports their dangers.

The following code is a minimal example showing that Common Lisp always performs clean-up even when processing fails midway. Even if an error occurs during parsing, the temporary buffer is always released and the count of allocated buffers returns to 0.

Code 3. Common Lisp: guaranteed clean-up with unwind-protect `l1_unwind.lisp`

```lisp
;;; unwind-protect: cleanup guaranteed by the language (teaching example)
(defvar *open-buffers* 0)                ; buffers currently held (for observation)

(defun allocate-temporary-buffer ()
  (incf *open-buffers*)
  (make-array 16 :element-type '(unsigned-byte 8)))

(defun free-temporary-buffer (buffer)
  (declare (ignore buffer))
  (decf *open-buffers*))

(defun parse-multipart-data (request buffer)
  (declare (ignore buffer))
  (when (eq request :malformed)          ; simulate failure mid-parse
    (error "budget exceeded while parsing"))
  :parsed)

(defun process-web-request (request)
  (let ((buffer (allocate-temporary-buffer)))
    (unwind-protect
         (parse-multipart-data request buffer)  ; protected body
      (free-temporary-buffer buffer))))         ; runs on normal exit and on any non-local exit

(print (list (process-web-request :ok) *open-buffers*))
(print (list (handler-case (process-web-request :malformed)
               (error () :rejected))            ; caller turns the error into a rejection
             *open-buffers*))
```

Output (GNU CLISP 2.49.95+, -ansi)

```
(:PARSED 0)
(:REJECTED 0)
```

For policymakers

C's `longjmp` is like rushing out through the emergency exit during a fire without tidying up. Common Lisp's `unwind-protect` corresponds to a building in which, whichever exit you leave by, a clean-up crew always does its job just before the door. Because clean-up is guaranteed as a rule of the language, a "half-wrecked room" is never handed to the next occupant. Standard file handling (`with-open-file`) uses the same mechanism and closes the file whatever happens along the way (Appendix A.3).

8.2 Why PHP continued to be implemented in C

There were nevertheless historical and practical reasons why PHP has been implemented in C, in the Zend Engine. First, in the late 1990s and early 2000s, when PHP spread, Common Lisp runtimes consumed a great deal of memory, a disadvantage for handling many requests on cheap shared hosting. Under CGI, which starts a process for each request, small C programs that started quickly had the advantage. Second, PHP's strength lay in its ability to incorporate, as extensions, the huge body of existing C libraries such as MySQL, OpenSSL and libxml2. Third, the Zend Engine implemented in C a fast memory manager specialised for short-lived processing, discarding memory wholesale at the end of each request. Ironically, the combination of that simplicity with workers that live on across requests became the breeding ground for these defects.

A comparison should include compatibility with PHP, replacements for extensions, pause times due to garbage collection, concurrency performance, operating staff and the cost of verification. Implementations that run PHP on .NET, such as PeachPie, are examples of alternative runtimes, but they require compatibility assessment and review of native dependencies, and cannot be said to be free of every form of remote code execution. [44] Short of replacing the language, an intermediate measure is to move dangerous native processing into a separate process.

The author's government R&D: Lispache and Bullet Proof Server

From 1998 to 2003, in research and development funded by the government budget of the Ministry of International Trade and Industry (MITI), the author used Franz Inc.'s Allegro Common Lisp to develop web, mail and DNS servers, among others, called Lispache and Bullet Proof Server. This experience is the practical background of the idea discussed in this paper: designing the resource management of the language implementation and the fault tolerance of the whole system as one.

Lispache was a web server upwardly compatible with Apache. On an ordinary web server, the slash-separated path after `CyberAcademy/` in a URL such as `https://crl.co.jp/CyberAcademy/` exposes the directory structure under the UNIX `public_html` directory as it is. HTTP thus carries the risk of revealing the internal file structure to the outside. To reduce this risk, Lispache had no actual directory structure such as `CyberAcademy/…/…/` , and no files such as `index.html` existed either. Every element was stored as an object in an encrypted object database, and on each GET request the server generated the response in real time, as if the directories and files existed. Even if UNIX root privileges were seized, an attacker would therefore find only huge encrypted object store files, and could not directly alter or steal their contents.

Bullet Proof Server extended the same approach to other web services. The servers holding designers' work and other data were physically disconnected from the network; content was transferred to the Internet-facing servers, encrypted and over a separate path, only at update time. In addition, a separate monitor server checked that the content of the Internet-facing servers was identical to that of the content-providing servers. Thanks to Lisp's dynamic nature, maintenance and version upgrades were carried out with no downtime at all. The system ran without a single stop for about ten years after it went into operation, until it had to be shut down when the building was rebuilt.

CRL's official list of government projects includes, among others, the Ministry of International Trade and Industry projects "Development of an advanced environment for distribution, supply and reuse enabling usage-based charging" (MMCA-2005) and "Development of a next-generation dynamic distributed shared VM home server system" (No. 11 Jōgi-Ō 1317). The English version also describes research and development on a next-generation dynamic object-oriented secure computing architecture, a content distribution environment and a distributed shared dynamic VM home server. [48,49]

What this paper takes from these experiences is the importance of combining memory management, handling of abnormal conditions and the demarcation of responsibility between services from the design stage. Today's DevSecOps policy connects this design principle to continuous verification, control of privileges, and the operation of updates and recovery.

8.3 NASA Remote Agent: combining language, verification and diagnosis

On NASA's Deep Space 1 probe, the flight experiment of the autonomous control software Remote Agent was conducted in the week of 17 May 1999, and JPL announced on 7 June that all objectives had been achieved. [31] Its Executive was written in multithreaded Lisp. During the flight, a process waiting for an event stopped making progress. According to NASA's post-flight analysis, the cause was a race condition between two threads due to a missing critical section around a condition wait, and it was of the same form as an error found in another subsystem by model checking (SPIN) before flight in 1997. The error had been reintroduced into a sibling subsystem that had not been covered by pre-flight verification. [30] According to developers' recollections, the ground team obtained a backtrace from the spacecraft, identified the cause, and resumed processing by manually generating the awaited event. The cause lay in a call to a low-level function that bypassed the safety guarantees of the team's custom-designed language. [65,66]

The lesson to draw from this case is not that "accidents cannot happen because it is Lisp". It is, first, that the combination of safe abstraction (the language), formal verification, the ability to diagnose a running system and ground operations capability can support a highly reliable system; and second, that low-level calls bypassing safe abstractions bring danger outside the scope of verification. The latter applies equally to Rust's `unsafe` and to the FFI of every language.

9 Isolation, restarts and serverless in practice

Containers normally share the host kernel. Namespaces, resource control (cgroups), running as non-root, dropping privileged capabilities, restricting system calls (seccomp) and mandatory access control (MAC) are combined to limit the spread of a compromise. Where sensitivity is high or attackers may share the same machine, virtual machines, microVMs or dedicated hosts should be chosen. Ordinary operating-system processes are also separated by virtual memory; the claim that "before containers, one crashing C program always brought down the whole server" is wrong. [34,35,36]

Kubernetes restarts and rescheduling are mechanisms for availability, not complete removal of a compromise. Redeploying the same vulnerable image invites another intrusion. Stolen credentials, altered databases, persistent volumes, and compromised nodes or build infrastructure (CI) require separate treatment. Recovery time depends on monitoring intervals, restart back-off, scheduling and start-up processing; there is no uniform guarantee such as “within one second”.

AWS documentation states explicitly that Lambda execution environments are reused, and that objects declared outside the function handler and the contents of `/tmp` can persist across invocations. [32] It therefore cannot be said that “with serverless, all memory disappears every time, so UAF is impossible”. Firecracker is a small virtual machine monitor using Rust and KVM that keeps the attack surface small, but it still requires verification that covers the host kernel, its `unsafe` parts, and configuration and operations. [33] None of these mechanisms can undo information theft completed while the first request was being processed.

Layer	Role	Main means	What it cannot prevent alone
Code	Do not create defects	Memory-safe languages, review, static analysis	Design and authorisation errors, `unsafe`/FFI
Runtime	Do not carry broken state forward	`unwind-protect`, `Drop`, process isolation, periodic replacement of workers	Theft completed during processing; defects in the runtime itself
Platform	Confine damage to a compartment	Reduced container privileges, microVMs, network segmentation	Defects in a shared kernel; excessive privilege settings
Operations	Fix fast, notice fast, recover reliably	Asset register, automated deployment, detection, preservation of evidence, recovery drills	Assets missing from the register; procedures never rehearsed

10 Making DevSecOps an operating model and an architecture

10.1 What DevSecOps is

DevSecOps links development, security and operations through the same change management and the same chain of evidence. It turns security from “an inspection just before release” into “an automated, repeated activity built into every step from development to operations”. CI stands for continuous integration (merging changes frequently and building and testing automatically each time); CD stands for continuous delivery or deployment (keeping verified artefacts ready to be deployed safely at any time). NIST’s Secure Software Development Framework (SSDF, SP 800-218) provides a framework for building secure practices into existing development processes, and a draft revision (v1.2) was published in December 2025. [37,80] DevSecOps is neither a product name nor a collection of scanning tools; it is a continuous way of working with responsibility and evidence attached.

For policymakers: how old buildings stay safely in use

Not every old building can be rebuilt. Lives are protected nonetheless because building codes and fire services exist. DevSecOps can be seen as the same mechanism for software.

Building safety	Counterpart in information systems (DevSecOps)
Building register and drawings	Asset register and software bill of materials (SBOM): what runs where
Building permits and completion inspection	Automated build, test and inspection gates (CI/CD)
Fire compartments and fire doors	Least privilege and network segmentation: an intrusion does not spread
Fire alarms and sprinklers	Behavioural detection and automatic containment
Periodic inspection and repair	Matching against vulnerability information and rapid deployment of fixes
Evacuation drills	Exercises assuming compromise, and recovery drills
Fire law and inspections	Government mandates, procurement conditions and independent assessment

10.2 The closed loop governments should institutionalise

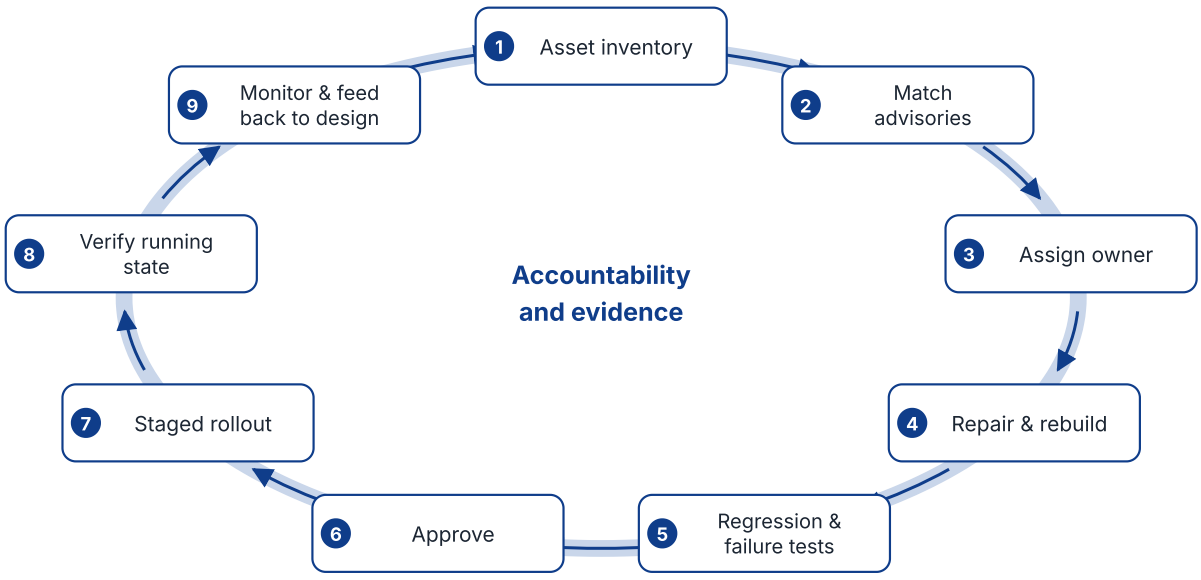

Figure 2. The DevSecOps closed loop governments should institutionalise

The closed loop that governments should institutionalise is the cycle: asset register → matching against vulnerability information → assignment of an owner → fix and rebuild → regression and failure-path testing → approval → staged deployment → verification of the running state → feeding monitoring results back into design. Publication of vulnerability information does not by itself complete a safe transition. What matters is assigning an owner and records to each stage and making visible where things are stalled.

Configuration is managed as code, and approved artefacts are pinned by digest (a unique identifier computed from their content). Rather than patching running servers by hand, artefacts incorporating the fix are built from a clean state every time, and the very same artefact that was verified is deployed to production (immutable infrastructure). The software bill of materials (SBOM) and build provenance are retained, and changes to suppliers, dependencies and configuration are tracked. A digital signature is evidence of "who released what"; it does not prove that the signed code is safe. The build infrastructure itself is protected with least privilege, short-lived credentials and independent approval.

Trust is not placed in network location alone; authentication and authorisation are performed for each service. The privileges of a public web service must not give direct reach to deleting backups, extracting a whole database, cloud management functions or the authentication infrastructure. [38] Using a web server as a foothold to move laterally to the authentication infrastructure or backups is a typical post-intrusion technique. Breaking this chain of privileges, before replacing any product, is the reform with the most immediate effect.

10.3 Mapping stages to controls

Stage	Purpose	Main practices
Plan and implement	Do not create defects (shift left)	Threat modelling, adoption of memory-safe languages, review, static analysis (SAST), sanitizers
Build and test	Stop known dangers before production	Rebuilds from clean infrastructure, scanning of dependencies and images, fuzzing, failure-path testing, recording of provenance
Deploy and run	Confine damage to a compartment even when an unknown defect is exploited	Least privilege, network segmentation, staged deployment, replacement of workers and containers
Monitor and respond	Notice fast, keep evidence, recover reliably	Observability, behavioural detection, preservation of evidence, revocation of credentials, recovery drills, feedback into design

10.4 What would have differed on the day of PHP2Shell (illustration)

The following is an **illustrative scenario** showing how the response to the same incident changes with and without the capabilities described in this paper. It does not describe measurements at any particular organisation.

Phase	Organisation without controls	Organisation with DevSecOps in place
Advance notice of disclosure	Begins to identify systems that use PHP; inquiries to contractors take time	Immediately extracts the affected systems, versions, fix history and owners from the asset register
Fixed release published	Fix procedures are manual and there is no test environment, so the service is stopped for the work	Automatically builds and scans images incorporating the fix and deploys them in stages, starting with a subset of users
A few hours later	Some servers remain unpatched; nobody knows how far the fix has reached	Confirms replacement of all workers and the running versions and settings; exceptions recorded with deadlines
If already intruded	The attacker reaches the authentication infrastructure and backups from the web server's privileges	Segmentation and least privilege keep the damage within the compartment; detection leads to containment and revocation of credentials
The following days	Prolonged outage; difficulty in being accountable	Reporting based on records; design changes to prevent recurrence

11 Measuring the effect: separating security from continuity

11.1 Harm is determined by multiplication

Define the unit of evaluation as, for example, “an event in which compromise of a public web worker leads to unauthorised modification of a sensitive database”. For this specific path, let E be exposure to dangerous input, X the success of compromise and L reaching additional critical assets. Then the following decomposition holds.

$$P(E \cap X \cap L) = P(E) \times P(X \mid E) \times P(L \mid E \cap X)$$

This formula does not assume independence of the events; it is a chain of conditional probabilities. Constraining input and disabling unnecessary functions suppress exposure; rapid fixes and safe implementation suppress the success of compromise; least privilege and segmentation suppress reaching later stages. The formula is an explanatory decomposition, not a guarantee covering every attack path. The value of the information held by the first compromised worker itself must also be assessed separately.

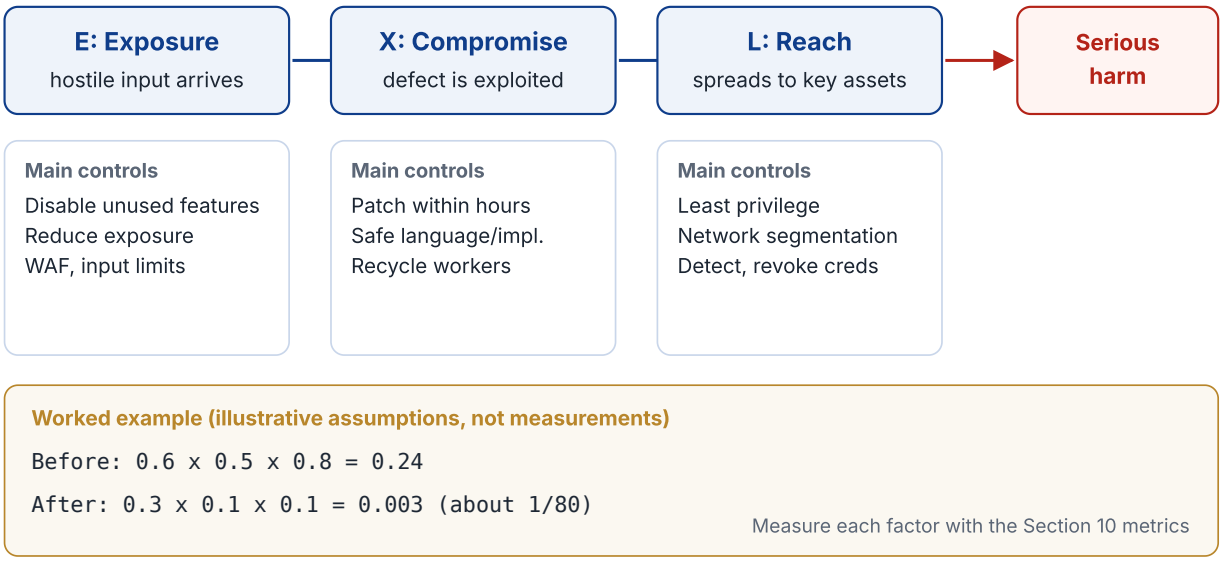

Figure 3. The attack chain and the controls that suppress each stage

For policymakers: a numerical example (assumed)

The figures below are **assumptions** to explain the mechanism, not measured values. Suppose that for a public service the probability that an attackable function is exposed is 0.6, the probability that an attack succeeds because the fix is delayed by a week after disclosure is 0.5, and the probability of reaching the authentication infrastructure or database after intrusion is 0.8. The overall probability is then $0.6 \times 0.5 \times 0.8 = 0.24$. If DevSecOps lowers exposure to 0.3 by disabling unnecessary functions, the probability of success to 0.1 by deploying the fix within hours, and the probability of reach to 0.1 through least privilege and segmentation, the overall figure becomes $0.3 \times 0.1 \times 0.1 = 0.003$, roughly one-eightieth. Even if no single measure is perfect, stacking layers that act at different stages lowers the overall risk greatly. How much each layer actually contributes is measured with the metrics below.

11.2 Metrics to measure

Metric	Definition	Factor
Public asset coverage	Share of public assets found by external observation that are registered in the asset register with an owner	E
End-of-life assets	Number of systems containing runtimes, operating systems or components past end of support, and their migration deadlines	X
Time to deploy fixes	Time from disclosure of a critical vulnerability to completion of deployment to all targets and verification of the running state (median and 95th percentile)	X
Privilege escalation blocking rate	Share of compromise-assumed tests in which reach from web worker privileges to critical assets was blocked	L
Time to detect and contain	Time from the appearance of signs of compromise to detection, isolation and revocation of credentials	L
Recovery test success rate	Share of planned recovery drills that met the recovery time objective (RTO) and recovery point objective (RPO)	Continuity
Change failure rate	Share of deployed changes that required remediation or rollback	Continuity

Targets are set according to the criticality of each business function. Comparisons are made for the same group of services before and after adoption, recording load, exposure and frequency of change. The share of simulated attacks that were blocked must not be equated with the actual probability of compromise. A decline in the number of findings from scanning tools alone cannot rule out an apparent improvement from a narrower scan scope or registered exceptions. Recovery drills should verify not only restoration from backup but also revocation of keys, prevention of recurrence and data integrity.

On the relationship between DevSecOps capabilities and operational performance, there is the continuing research of DORA (DevOps Research and Assessment). Its 2021 report found that the elite group had both lead time for changes and time to restore service of less than one hour. [76] This is a correlational survey based on

self-reporting and does not directly prove security benefits, but it shows that the capability to “deploy fixes safely within hours” is achieved by real organisations.

12 A mandate model and implementation sequence for governments

The following is a **policy proposal** to be adapted to each country’s executive orders, procurement conditions and supervisory systems; it is not a common law already in force in every country. What is mandated is not a particular cloud or language but controls and responsibilities that can be demonstrated continuously.

1. **Assets and accountability:** Register running operating systems, language runtimes, extensions, dependencies, exposure paths and owners, and require contractors to submit the same. Record package fix history, not just version numbers.
2. **Updatability:** Make support periods, emergency contacts, provision of fixes, update testing and the capability to restart and deploy conditions of contract.
3. **Evidence of change:** Record reviews, reproducible-build targets, bills of materials, provenance, signature verification and the digests of running artefacts.
4. **Privilege and isolation:** Separate web, database, management functions and backups, and restrict permitted communications. Give each exception an owner and an expiry date.
5. **Continuous verification:** Combine static and dynamic analysis, dependency scanning, fuzzing, and authorisation and failure-path testing according to risk.
6. **Incident response:** Define and rehearse roles and decision criteria for monitoring, preservation of evidence, containment, credential renewal, recovery and external reporting.
7. **Data protection:** Limit the purpose, retention period and access rights of logs, and avoid indiscriminate collection of secrets and personal data.
8. **Language migration:** Require memory-safety roadmaps that give priority to processing of dangerous input, privileged components and native dependencies. [39,71]
9. **AI-generated code:** Require an accountable human owner, independent review, isolated testing, and screening for licences and secrets.
10. **Funding and oversight:** Budget for shared testing facilities for small suppliers, maintenance of foundational open-source software, independent assessment and periodic review of the regime.

12.1 Implementation sequence

Period (proposed)	Priority actions	Evidence of results
Immediately to 30 days	Inspect public assets such as PHP, apply fixes, eliminate old workers, restrict dangerous public functions	Running state and owners are known; emergency exceptions have deadlines
1–3 months	Implement protected CI/CD, least privilege, monitoring and recovery drills for representative services	Records of compromise-assumed tests and recovery tests exist
3–12 months	Reusable government-wide platforms, renewal of contracts, start of replacement of high-risk components	Time to deploy fixes and reachable privileges improve as measured
12 months onward	Extend verified migration; reassess technical debt and supplier concentration	Improvements in continuity and security are assessed independently

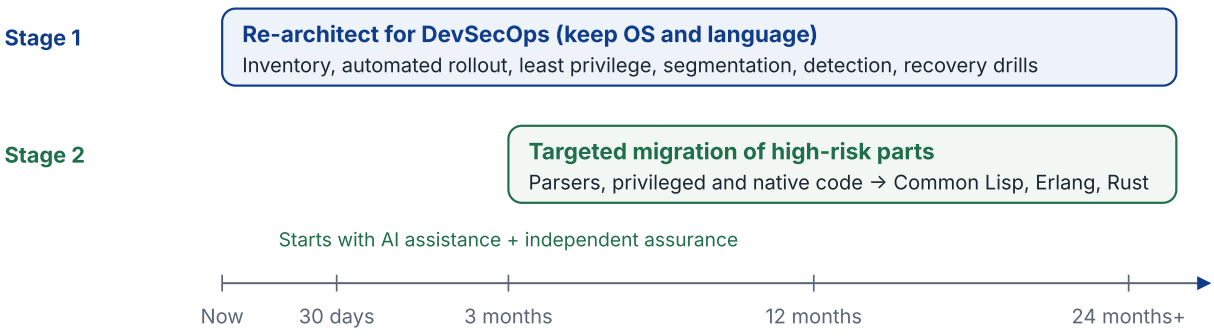

Figure 4. The two-stage policy timeline (indicative)

12.2 A 72-hour action plan after disclosure

A standard action plan for ministries and local governments to follow when a critical vulnerability in foundational software such as PHP2Shell is disclosed should be defined in peacetime.

Time	Action
Advance notice to hour 0	Extract affected systems from the asset register and convene owners. Enable interim defences such as WAF rules. Confirm the criteria for shutdown (exposure, information handled, whether a fix is possible)
Hours 0–6	Build and scan artefacts incorporating the fixed release, and deploy them in stages starting with the most critical public services. For anything that cannot be fixed, choose reduced functionality, access restriction or shutdown

Time	Action
Hours 6–24	Verify running versions and settings of all targets. Preserve logs from before and after disclosure and investigate signs of compromise. Revoke credentials where necessary
Hours 24–72	Record deadlines and alternatives for exceptions. Explain to the public, distinguishing vulnerability, attempt, compromise and shutdown. Incorporate additional advisories (such as that of 23 October in this incident)

In fields such as health care, transport and defence, simultaneous automatic updates can cause other safety incidents. Design for limited advance deployment, alternatives when functions are stopped, and recovery to a known-safe configuration. Avoid, however, unconditional rollback to a vulnerable old version; combine it with isolation or reduced functionality.

12.3 Alignment with existing frameworks

The proposals in this paper can be introduced in alignment with frameworks already advancing in each country.

Country or region	Existing framework	Relation to this paper's proposals
United States	NIST SSDF (SP 800-218); BOD 22-01, which sets deadlines for remediating known exploited vulnerabilities (KEV), generally two weeks; calls for memory-safety roadmaps [37,39,71,72]	Build the time-to-deploy-fixes metric and language migration plans into procurement conditions
European Union	Cyber Resilience Act (Regulation 2024/2847). From 11 September 2026, manufacturers must submit, for actively exploited vulnerabilities, an early warning within 24 hours, a notification within 72 hours and a final report within 14 days after a corrective measure is available [73]	PHP2Shell is one of the first large-scale incidents since the reporting obligations began to apply; without an asset register and bills of materials, reporting within the deadlines is difficult
Japan	Establishment of the National Cybersecurity Office in July 2025; entry into force on 1 October 2026 of the main parts of the so-called Active Cyber Defence Act (Act No. 42 of 2025), including reporting by operators of critical infrastructure [74]. On 24 October, a VPS provider designated an attack on its service the "first activation" case of active cyber defence and asked the government to act against the attack infrastructure [82,89]	Establish common standards for asset registers, time to deploy fixes and recovery drills across government agencies, local governments and critical infrastructure. Action against attack infrastructure does not replace the victim organisation's own capability to contain and recover

12.4 International coordination and investment in foundational software

In this incident, information leaked before the fix was announced, and incorrect information and attack code circulated. [52,55] Governments should establish internationally the arrangements that support coordinated disclosure of critical vulnerabilities, procedures for delivering fixes to critical infrastructure before publication,

and common mitigations that can be consulted immediately after disclosure. According to one security researcher's account, signs of this problem had already been found in an external audit in 2024, but work stalled because agreement on how to fix it proved difficult. [52] Sustained public funding for the maintenance and auditing of foundational software on which public administrations worldwide depend is a rational investment in national security.

13 Using AI to carry out replacement safely

AI can be used as an aid in translating code written in C and other languages into Common Lisp, Erlang or Rust. But successful compilation does not guarantee that meaning is preserved. Integer widths, signedness, character encodings, time, concurrency, abnormal termination, protocol compatibility and the semantics of database transactions can all change. Relying only on comparison tests against the old implementation freezes the old implementation's defects as the "correct answer".

First, humans define the input and output specifications, privileges, resource limits and error contracts, and test with both normal and malicious input. Property-based testing, fuzzing, formal verification of state machines and critical invariants, and independent review are used together. AI-generated code must not be allowed to add raw pointer manipulation, `unsafe`, FFI, dynamic evaluation or execution of external commands without review.

Replace in small steps, starting from high-risk boundaries, and compare results by running alongside the existing service. Do not give generative AI production secrets or deployment credentials, and separate the proposal of changes from approval for production. Choose the language not only for its technical properties but also for availability of maintainers, supply of libraries, performance targets and ease of verification. There is no need to call Common Lisp and Erlang simply "new languages": what matters is not their age but safe abstraction and operability.

In this incident, AI-assisted analysis was used in part of the discovery of the defects. [52,55] On the attack side, signs of an AI agent in post-intrusion activity at a European government entity were reported, but the report was withdrawn because initial access through PHP2Shell could not be confirmed. [53] Even so, the same capability that accelerates discovery is available to attackers. As the speed of attack rises, defenders too must use AI-assisted scanning, translation and test generation within a framework of responsibility and verification.

14 Conclusion: institutionalise the ability to change safely, without waiting for wholesale replacement

This problem should not be reduced to a binary choice between "dangerous because the language is old" and "safe once it is in a container". The societal losses arose from the combination of a defect in the foundation, the lifetime of request state, execution privileges, where secrets were kept, delays in fixing and, not least, not knowing one's own assets.

The first task of government is to build a DevSecOps capability that, even for services running on existing Linux and UNIX-family operating systems and PHP and the like, knows its assets, deploys fixes reliably and quickly, narrows the reach of a compromise and verifies continuity. This can be done without changing operating systems or languages, and it reduces the probability and scale of harm multiplicatively. The second task is to advance, with AI as an aid, a focused migration of high-risk components that handle external input to safe languages and runtimes, reducing the reproduction of the same kinds of defects.

The two must proceed in parallel, linking budgets, responsibilities and verification results. What should be guaranteed is not a declaration that “we will never be compromised”, but the capability to reduce risk continuously and evidence that can account for the results.

Appendix A. Reading the defensive code (all examples executed)

The code in this appendix is for teaching purposes; it is neither the actual PHP fix nor attack code. Memory exhaustion is simulated as a logical budget, not real OOM handling. All code was compiled and run in the author's environment (Ubuntu 24.04 LTS) with the following toolchains, producing the output shown: gcc 13.3.0 (AddressSanitizer), rustc 1.97.0, GNU CLISP 2.49.95+ (with `-ansi`) Erlang/OTP 28 and PowerShell 7.4.6 (Linux build). YAML syntax was checked with PyYAML, and shell scripts with `bash -n`. For production use, test with the toolchain you adopt and additionally verify time limits, I/O, concurrency and external side effects.

A.1 C and Rust: how the same mistake is treated (Codes 1 and 2)

Location	Meaning
Code 1 (1)	`malloc(16)` borrows 16 bytes and keeps the address in `session`.
Code 1 (2)	`free(session)` returns the memory. Yet the variable `session` still holds the old address, and the language does not invalidate it.
Code 1 (3)	Reads memory that has been returned. This is undefined behaviour under the standard, and compilers do not normally reject it. Only with AddressSanitizer is it reported at run time as a "heap-use-after-free".
Code 2 (1)–(2)	In Rust, the owner of the `String` is the variable `session`. `drop(session)` moves ownership into the function, and the memory is freed.
Code 2 (3)	Use of a variable that has lost ownership is rejected by the compiler's borrow checker (error E0382). No executable is produced.

Policy implication: In C, analysis tools can find such errors, but only when tests exercise the faulty path. In Rust, this kind of error is structurally excluded within safe code. Procurement can set conditions on both: at which stage analysis tools must always be used, and in which language new development is done.

A.2 Common Lisp: unwind-protect (Code 3)

Location	Meaning
`*open-buffers*`	An observation variable that counts temporary buffers currently held. It returns to 0 if clean-up takes place.

Location	Meaning
`parse-multipart-data`	If the request is `:malformed`, it signals an error to simulate failure partway through parsing.
`unwind-protect`	Whether the first argument (the body) finishes normally or is exited by an error or other means, the remaining arguments (the clean-up) are always executed.
`handler-case`	Catches the error at the upper level and converts it into a rejection of the request (`:rejected`).
Output	`(:PARSED 0)` and `(:REJECTED 0)`. On success and on failure alike, the count of held buffers has returned to 0.

A.3 Common Lisp: with-open-file (Code 4)

The standard macro `with-open-file` always closes the file when the body is exited. On an abnormal exit, it closes the stream with `:abort t`. [78] The expansion shown in the comments illustrates the concept; the actual expansion differs between implementations.

Code 4. Common Lisp: with-open-file and its conceptual expansion `l2_open_file.lisp`

```lisp
;;; with-open-file: a standard macro that always closes the file
(defvar *seen* nil)                       ; remember the stream to inspect it later

(defun read-first-line (path)
  (with-open-file (stream path :direction :input)
    (setf *seen* stream)
    (read-line stream)))

;; Conceptual expansion (details differ between implementations)
;; (let ((stream (open path :direction :input)) (abort t))
;;   (unwind-protect
;;        (multiple-value-prog1 (progn (setf *seen* stream) (read-line stream))
;;          (setq abort nil))                ; marks a normal exit from the body
;;     (when stream (close stream :abort abort))))  ; abnormal exit closes with :abort t

(defun demo (path)
  (with-open-file (out path :direction :output :if-exists :supersede)
    (write-line "first line" out))
  (print (read-first-line path))          ; => "first line"
  (print (open-stream-p *seen*))          ; => NIL (closed)
  (print (handler-case
             (with-open-file (stream path)
               (setf *seen* stream)
               (error "failure inside the body"))
           (error () :handled)))          ; => :HANDLED
  (print (open-stream-p *seen*)))         ; => NIL (closed after an error too)
```

Output (GNU CLISP 2.49.95+)

```
"first line"
NIL
:HANDLED
NIL
```

Output	Meaning
`"first line"`	The first line of the file was read.
`NIL` (line 2)	After the body exited normally, the stream is closed.
`:HANDLED`	An error occurred in the middle of the body and was handled at the upper level.
`NIL` (line 4)	Even after exiting because of an error, the stream is closed.

A.4 Common Lisp: the condition system and restarts (Code 5)

In Common Lisp's condition system, the lower-level function that detects an error defines "recovery options" (restarts), and an upper-level handler examines the situation without unwinding the stack and then decides which option to use. Here, a logical budget overrun occurs while parsing chunk 3 (a fragment of the received data), and depending on policy the handler chooses either "free the cache and retry" or "reject the whole request".

Code 5. Common Lisp: the upper level sets policy; recovery happens where the problem occurred `l3_restart.lisp`

```
;;; Conditions and restarts: the caller chooses the policy, the callee carries it out
;;; (a simulation of a logical memory budget, NOT a real OOM handler)
(define-condition memory-limit-exceeded (error)
  ((requested-size :initarg :size :reader requested-size))
  (:report (lambda (condition stream)
             (format stream "Memory limit exceeded while requesting ~A bytes."
                     (requested-size condition)))))

(defvar *cache-bytes*)                    ; cache that can be released (logical units)

(defun parse-chunk (chunk-id)
  (let ((retries 0))
    (loop
      (format t "[Parser] Processing chunk ~A...~%" chunk-id)
      (restart-case
          (progn
            (when (and (= chunk-id 3) (> *cache-bytes* 0))
              (error 'memory-limit-exceeded :size 1024))  ; signalled where it happens
            (return :ok))                                 ; leave the loop on success
        (free-cache-and-retry ()                ; option 1: change state, then retry
          :report "Free the cache and retry this chunk."
```

```
          :test (lambda (c) (declare (ignore c)) (< retries 1)) ; at most once
          (incf retries)
          (format t "[Restart] Freeing ~A bytes of cache.~%" *cache-bytes*)
          (setf *cache-bytes* 0))))))           ; change state before retrying

(defun process-request (n-chunks)
  (restart-case
      (progn
        (dotimes (i n-chunks)
          (parse-chunk (1+ i)))
        :accepted)
    (reject-request ()                          ; option 2: reject the whole request
      :report "Reject the whole request."
      (format t "[Restart] Request rejected; no partial result is kept.~%")
      :rejected)))

(defun run-with-policy (policy &key (cache 4096))
  (let ((*cache-bytes* cache))
    (handler-bind                               ; caller: decides the policy
        ((memory-limit-exceeded
           (lambda (c)
             (format t "[Handler] ~A Policy: ~A~%" c policy)
             (let ((retry (find-restart 'free-cache-and-retry c)))
               (if (and (eq policy :retry) retry)
                   (invoke-restart retry)
                   (invoke-restart 'reject-request))))))
      (process-request 5))))
```

Output: (run-with-policy :retry) and (run-with-policy :reject)

```
[Parser] Processing chunk 1...
[Parser] Processing chunk 2...
[Parser] Processing chunk 3...
[Handler] Memory limit exceeded while requesting 1024 bytes. Policy: RETRY
[Restart] Freeing 4096 bytes of cache.
[Parser] Processing chunk 3...
[Parser] Processing chunk 4...
[Parser] Processing chunk 5...
=> :ACCEPTED

[Parser] Processing chunk 1...
[Parser] Processing chunk 2...
[Parser] Processing chunk 3...
[Handler] Memory limit exceeded while requesting 1024 bytes. Policy: REJECT
[Restart] Request rejected; no partial result is kept.
=> :REJECTED
```

Location	Meaning
`define-condition`	Defines a kind of condition, “memory limit exceeded”. It holds the requested amount and generates a readable description.

Location	Meaning
`restart-case` (in `parse-chunk`)	Defines the recovery option `free-cache-and-retry`, usable where the problem occurred.
`:test`	Enables this option only while fewer than one retry has been made: a limit that prevents endless retrying.
`(setf *cache-bytes* 0)`	Changes the state (frees the cache) before retrying. Retrying without changing the state would only repeat the same failure.
`reject-request`	The option of rejecting the whole request. Nothing parsed partway is left behind as if it were a “normal result”.
`handler-bind`	The upper-level handler. It is called before any unwinding and invokes an option based on policy and on whether a retry is available (`find-restart`).
Output (:retry)	Failure at chunk 3 → handler → cache freed → chunk 3 reprocessed successfully → chunks 4 and 5 follow → `:ACCEPTED`.
Output (:reject)	Failure at chunk 3 → handler → request rejected → `:REJECTED`. Chunks 4 and 5 are not processed.

Technical note: a retry needs a change of state and a limit

The author’s original notes used an example in which a restart simply called the same function again recursively. When that version was run on GNU CLISP, the cache state did not change, so the same condition was signalled repeatedly, and the program stopped with a stack overflow after about 3,500 lines of output. Code 5 was revised to change the state before retrying, to cap the number of retries, and to reject the whole request when recovery is not possible. A design that skips part of the received data and makes the transaction look normal is not adopted, because it invites tampering and inconsistency.

A.5 Rust: returning a budget overrun as an ordinary error (Code 6)

Code 6. Rust: limit checks and fallible allocation `r1_bounded.rs`

```rust
use std::collections::TryReserveError;

#[derive(Debug)]
enum InputError {
    TooLarge,                       // over the limit: reject the request
    Capacity(TryReserveError),      // allocation failed: return an error, do not abort
}

fn copy_bounded(input: &[u8], limit: usize) -> Result<Vec<u8>, InputError> {
    if input.len() > limit {
        return Err(InputError::TooLarge);       // check before allocating
    }
    let mut owned = Vec::new();
    owned
        .try_reserve_exact(input.len())         // fallible allocation API
        .map_err(InputError::Capacity)?;
    owned.extend_from_slice(input);
    Ok(owned)
}

fn main() {
    for (input, limit) in [(&b"abcd"[..], 3), (&b"abc"[..], 3)] {
        match copy_bounded(input, limit) {
            Ok(v) => println!("accepted: {} bytes", v.len()),
            Err(InputError::TooLarge) => println!("rejected: too large"),
            Err(InputError::Capacity(e)) => println!("rejected: {e}"),
        }
    }
}
```

Output (rustc 1.97.0)

```
rejected: too large
accepted: 3 bytes
```

Location	Meaning
`enum InputError`	Enumerates the kinds of failure as a type. The compiler requires callers to handle every case.
`input.len() > limit`	Compares the size of the input against the limit before allocating memory.
`try_reserve_exact`	An API that returns an error, rather than aborting the process, when allocation fails.
`?`	On error, returns immediately to the caller.

In this example the input itself is already in memory before copying, so this is no substitute for streaming limits at the network entrance. Nor does `try_reserve` prevent forced termination by the operating system at a later stage. [28,29]

A.6 Rust: when clean-up (Drop) runs and when it does not (Code 7)

Code 7. Rust: clean-up through Drop and the panic setting — r2_drop.rs

```rust
struct Parser {
    name: &'static str,
}

impl Drop for Parser {                // cleanup that runs when the owner goes away
    fn drop(&mut self) {
        println!("cleanup: {}", self.name);
    }
}

fn parse(fail: bool) -> Result<(), String> {
    let _p = Parser { name: "multipart" };
    if fail {
        return Err("budget exceeded".to_string()); // Drop still runs on early return
    }
    Ok(())
}

fn main() {
    println!("{:?}", parse(true));

    let r = std::panic::catch_unwind(|| {         // with panic=unwind
        let _p = Parser { name: "during-panic" };
        panic!("simulated fatal error");           // Drop runs during unwinding
    });
    println!("panic caught: {}", r.is_err());
}
```

Output (default panic=unwind)

```
cleanup: multipart
Err("budget exceeded")
cleanup: during-panic
panic caught: true
```

Output (compiled with -C panic=abort)

```
cleanup: multipart
Err("budget exceeded")
exit status: 134
```

Output	Meaning
`cleanup: multipart`	Even when the function was left by an early `return`, clean-up through `Drop` ran.
`cleanup: during-panic`	With the default setting, the stack was unwound on panic and `Drop` ran for the values along the way.
`exit status: 134`	With `panic=abort`, no unwinding took place and the process was aborted without printing `cleanup: during-panic` (134 indicates termination by SIGABRT).

Policy implication: It cannot be said that "with Rust, clean-up always happens". The guarantee changes with build settings and with forced termination by the operating system. Procurement specifications need to check not only the language name but also the build settings and the handling of abnormal termination.

A.7 Erlang: a parent observes the failure of a child process (Code 8)

Code 8. Erlang: a lightweight process per request, with monitoring `e1_monitor.erl`

```erlang
-module(request_demo).
-export([run/2]).

%% One lightweight process per request; the parent observes failure via 'DOWN'
run(Data, Limit) when is_binary(Data),
                      is_integer(Limit), Limit >= 0 ->
    Parent = self(),
    {Pid, Ref} = spawn_monitor(fun() ->
        case byte_size(Data) =< Limit of
            true  -> Parent ! {self(), accepted};
            false -> exit(request_too_large)      % only the child exits
        end
    end),
    receive
        {Pid, accepted} ->
            erlang:demonitor(Ref, [flush]),
            {ok, accepted};
        {'DOWN', Ref, process, Pid, Reason} ->    % the parent receives the reason
            {error, Reason}
    end.
```

Output: run(<<1,2,3>>, 2) and run(<<1,2>>, 2) (Erlang/OTP 28)

```
{error,request_too_large}
{ok,accepted}
```

Location	Meaning
`spawn_monitor`	Creates a lightweight process to handle the request and, at the same time, monitors its termination.
`exit(request_too_large)`	For a request that exceeds the limit, only the child process terminates, with a reason. Neither the parent nor other requests are affected.
`'DOWN'` message	The child's termination reaches the parent as a message that includes the reason. The parent can choose to restart or reject.
`demonitor(Ref, [flush])`	On normal completion, removes the monitor and discards any notification that arrives late.

This is not an implementation of an OTP supervisor, nor a test of real memory exhaustion. Automatically retrying operations such as payments additionally requires identifiers that prevent duplicate processing (idempotency keys) and transaction records. [26,27]

Appendix B. Deployment and operations templates for protecting existing PHP

The following templates show the intent of each setting. Replace placeholders such as SHAs and digests with reviewed values. Syntax was checked, but no integrated run tests were performed on GitHub, Kubernetes or Falco; this is not a complete production deployment. The scripts in B.7 and B.8 were run in a test environment and their output checked (conditions are given in each subsection).

B.1 PHP settings (Code 9)

Assuming that the fixed release has been applied and following the advisory of 23 October, the unnecessary upload progress feature is disabled. The numbers are illustrative and should be set for each service through load testing. `ini_set` within an application cannot protect parsing that takes place before the request starts. [1,22]

Code 9. php.ini: example settings to add to a patched deployment `b1_php.ini`

```
; Extra settings for an UPDATED deployment (not a fix on their own)
; Put them in the config of the SAPI actually running (e.g. PHP-FPM)
session.upload_progress.enabled = Off  ; per the 23 October advisory

post_max_size = 8M            ; POST body limit (illustrative)
upload_max_filesize = 2M      ; per-file upload limit (illustrative)
max_input_vars = 1000         ; number of input variables accepted
max_input_nesting_level = 32  ; nesting depth of input arrays
max_file_uploads = 4          ; files per request

; No universal memory_limit value is recommended here
; Restart all workers after changing startup settings
```

Setting	Meaning and caveats
`session.upload_progress.enabled`	Tracking of upload progress; enabled by default. Disabling it is advised until CVE-2026-115560 is fixed.
`post_max_size`, `upload_max_filesize`	Limits on the size of accepted bodies and files. Set them to the minimum the service requires.
`max_input_vars`, `max_input_nesting_level`	Limits on the number of input variables and their nesting depth. They curb parsing load.
Restart all workers	Startup settings take effect only once workers are replaced.

B.2 Limits at the web server (Code 10)

Code 10. nginx: limiting body size before handing requests to PHP `b2_nginx.conf`

```
# Limit body size before PHP sees it (one layer of defence in depth)
client_max_body_size    8m;     # larger requests get 413
client_body_buffer_size 128k;   # larger bodies go to a temp file
# Note: neither setting caps PHP's internal memory use
```

Requests exceeding `client_max_body_size` are rejected with error 413, and bodies exceeding `client_body_buffer_size` are written to a temporary file. [47] These are part of defence in depth; they are not limits on PHP's internal memory use.

B.3 Asset inventory (Code 11)

Code 11. Shell: a minimal inventory of PHP assets — b3_inventory.sh

```bash
#!/usr/bin/env bash
# Minimal PHP inventory. Record the results in the asset register
set -euo pipefail

echo "== Installed packages =="
if command -v dpkg-query >/dev/null; then
  dpkg-query -W -f='${db:Status-Abbrev} ${Package} ${Version}\n' 'php*' \
    2>/dev/null | awk '$1 == "ii" {print $2, $3}'   # installed only
elif command -v rpm >/dev/null; then
  rpm -qa 'php*' || true
fi

echo "== CLI version and setting (FPM/Apache may differ) =="
if command -v php >/dev/null; then
  php -r 'echo PHP_VERSION, " ", PHP_SAPI, PHP_EOL;'
  php -r 'echo "upload_progress=", ini_get("session.upload_progress.enabled"), PHP_EOL;'
fi

echo "== Running PHP processes =="
ps -eo pid,user,args | grep -E '[p]hp-fpm|[h]ttpd|[a]pache2' || true
```

Running this script in the author's environment displayed entries such as `php8.3-cli 8.3.6-0ubuntu0.24.04.13`. Version 8.3.6 looks older than the fixed 8.3.36, but the package changelog (`/usr/share/doc/php8.3-common/changelog.Debian.gz`) recorded the fixes for CVE-2026-109618 and 109619, dated 19 October 2026. [51] Use fix history, not version numbers, for the determination. CLI settings may also differ from those of PHP-FPM or Apache, so check each SAPI actually running.

B.4 A CI scanning gate (Code 12)

Do not reference third-party Actions by mutable tags (such as master); pin them to reviewed commits. Verify the scanning tools' versions, checksums and distribution channels as well. Do not use `ignore-unfixed`, which excludes all vulnerabilities without a fix in one sweep. Record the rationale, alternatives, owner and deadline for each necessary exception. [40,42]

Code 12. GitHub Actions: stop the pipeline when scanning fails b4_actions.yml

```yaml
name: Security gate
on:
  pull_request:
  push:
    branches: [main]
permissions:
  contents: read                    # restrict the default token to read-only
jobs:
  security-gate:
    runs-on: ubuntu-latest
    timeout-minutes: 20
    steps:
      - uses: actions/checkout@<REVIEWED_FULL_COMMIT_SHA>  # pinned, not a mutable tag
        with:
          persist-credentials: false
      # Insert a reviewed, version-pinned Trivy installation step here
      - name: Scan source and dependencies
        shell: bash
        run: |
          set -euo pipefail
          command -v trivy
          trivy fs --scanners vuln,secret,misconfig \
            --severity HIGH,CRITICAL --exit-code 1 .
      - name: Build and scan candidate
        shell: bash
        run: |
          set -euo pipefail
          docker build --pull -t service:candidate .
          trivy image --scanners vuln \
            --severity HIGH,CRITICAL --exit-code 1 service:candidate
      # A separate approved job promotes the tested digest to production
      # Pull-request jobs never receive production credentials
```

Location	Meaning and caveats
`permissions: contents: read`	Restricts the job's permissions to read-only.
`@<REVIEWED_FULL_COMMIT_SHA>`	Pins the external Action to a reviewed commit to prevent substitution.
`--exit-code 1`	Fails the job if critical or high vulnerabilities are found.
Required check	Exit code 1 alone does not block a merge. Make this job required through branch protection or rulesets, and limit unreviewed bypass permissions. [41]
Jobs on pull requests	Because a pull request's own Dockerfile can be used for an attack, run it in a disposable environment with no secrets.

Location	Meaning and caveats
Promotion	Promote to production, through an approved job, the very artefact with the scanned digest, not something rebuilt separately.

Defects not yet recorded in known-vulnerability databases, and misidentifications, remain, so a passing scan does not mean complete safety.

B.5 Restricting execution privileges and the network (Codes 13 and 14)

Verify the UID, writable locations, capacity and image against the application. Applying this as is to an existing image that does not support running as non-root can cause start-up failures. Use CPU and memory limits together with capacity planning. [35]

Code 13. Kubernetes: part of a Pod template `b5_pod.yaml`

```yaml
# Fragment of a Pod template (not a complete Deployment)
spec:
  automountServiceAccountToken: false   # no unneeded API credentials
  securityContext:
    runAsNonRoot: true                  # never run as root
    runAsUser: 10001
    runAsGroup: 10001
    fsGroup: 10001
  containers:
    - name: php-app
      image: registry.example.gov/service@sha256:<APPROVED_DIGEST>  # the approved artefact
      securityContext:
        allowPrivilegeEscalation: false # block privilege escalation
        readOnlyRootFilesystem: true    # hinders tampering and backdoors
        capabilities:
          drop: [ALL]                   # drop all capabilities
        seccompProfile:
          type: RuntimeDefault          # restrict system calls
      resources:
        requests:
          memory: 256Mi
          cpu: 250m
        limits:
          memory: 512Mi                 # memory ceiling for the whole container
          cpu: '1'
      volumeMounts:
        - name: scratch
          mountPath: /tmp
  volumes:
    - name: scratch
      emptyDir:
        sizeLimit: 64Mi
```

Code 14. Kubernetes: a default-deny network policy b6_netpol.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: public-services
spec:
  podSelector: {}                # applies to every Pod in the namespace
  policyTypes: [Ingress, Egress]  # deny ingress and egress by default
# Open only ingress, DNS and required APIs with separately reviewed allow rules
# Requires a network plugin that actually enforces NetworkPolicy
```

A network plug-in (CNI) that actually enforces NetworkPolicy is required; a declaration alone is not enforced. Allow only DNS, ingress and the necessary databases or APIs through separate rules. Validate in a test environment first, so that a bulk rollout without allow rules does not take public services down. [36]

B.6 Behavioural detection and safe response (Code 15)

Code 15. Falco: alert when a web worker spawns a shell directly b7_falco.yaml

```yaml
# Assumes Falco's standard spawned_process and container macros
- rule: Shell directly spawned by web worker
  desc: Investigate unexpected shell execution by a web worker
  condition: >
    spawned_process and container
    and proc.pname in (php-fpm, php-fpm8.2, php-fpm8.3,
                       php-fpm8.4, php-fpm8.5, apache2, httpd)
    and proc.name in (sh, bash, dash, zsh, ksh)
  output: >
    Unexpected shell parent=%proc.pname process=%proc.name
    container=%container.id user=%user.name
  priority: WARNING
  tags: [execution, web]
# Test real process names and legitimate child processes in your environment
# This rule alerts only; it does not block or delete anything
```

This rule depends on the name of the immediate parent process, so it misses attacks that do not spawn a shell or that go through a different parent. It may also fire on legitimate child processes. Falco itself does not block an attack before the UAF occurs. [43] Forwarding to a SIEM or similar requires configuring the destination, and installing an integration tool does not by itself complete safe automatic removal.

The response procedure is: record and correlate → isolate the network traffic of the suspect target → preserve evidence → revoke credentials → redeploy with patched artefacts → verify data integrity. Emergency blocking may sometimes take priority, but avoid loss of evidence or disruption of services through automatic deletion of everything. Tests that deliberately inject failures (chaos testing) should be done first in an isolated environment, and in production only within an authorised scope, with stop conditions and fallback services prepared.

B.7 Has the update reached the running processes? (Linux, Code 16)

As Section 6.3 explains, processes that are already running keep using the old executable or libraries after a package update. The following script checks, for each PHP-related process, whether its executable and the PHP libraries it has loaded have been deleted (replaced by an update), and marks those that need a restart as STALE.

Code 16. Shell: detect PHP-related processes still running old code — b9_running_php.sh

```
#!/usr/bin/env bash
# Check whether running PHP-related processes still use an old binary or
# PHP library that an update has replaced (read-only; run as root)
set -uo pipefail
pattern='^(php[0-9.]*|php-fpm[0-9.]*|php-cgi[0-9.]*|lsphp[0-9]*|frankenphp|apache2|httpd)$'

for dir in /proc/[0-9]*; do
  pid=${dir#/proc/}
  comm=$(cat "$dir/comm" 2>/dev/null) || continue
  [[ $comm =~ $pattern ]] || continue                    # PHP-related processes only
  [[ $pid == "$$" ]] && continue
  exe=$(readlink "$dir/exe" 2>/dev/null || echo '?')
  # Still mapping a deleted (i.e. replaced by an update) PHP library?
  stale_lib=$(grep -E '/(libphp|php)[^ ]*\.so[^ ]* \(deleted\)$' \
                "$dir/maps" 2>/dev/null | head -n 1)
  status=OK
  [[ $exe == *' (deleted)' || -n $stale_lib ]] && status=STALE  # needs a restart
  user=$(stat -c %U "$dir")
  printf '%-6s %-7s %-10s %-10s %s\n' "$status" "$pid" "$user" "$comm" "$exe"
done
```

Output (an update simulated in a test environment; the working directory path is shortened to "…")

```
STALE  3425    root       php8.3     …/php8.3 (deleted)
OK     3426    root       php        /usr/bin/php8.3
STALE  3427    root       apache2    …/apache2
```

Location	Meaning
`/proc/[0-9]*`	Examines every running process. Reading other users' processes requires root.
`pattern`	Targets the PHP CLI, PHP-FPM, php-cgi, LiteSpeed's lsphp, FrankenPHP and Apache. Apache is included because mod_php is loaded into Apache processes.
`readlink "$dir/exe"`	The executable actually running. If an update has replaced it, "(deleted)" is appended.
`(deleted)` in `maps`	Checks whether a loaded PHP library (such as mod_php's libphp) has been replaced.
`STALE` / `OK`	STALE needs a restart (worker replacement); OK is running the current files.

In the test, a copied PHP executable was started and then replaced on disk to simulate a package update (line 1). A test process started under the name apache2 loaded a library named libphp8.3.so, which was then replaced, to simulate the mod_php case (line 3). Its executable was not replaced, so its path shows no "(deleted)", but the library check marked it STALE. A process running the ordinary `/usr/bin/php` is OK (line 2). No run tests were performed with real Apache or PHP-FPM. Restart services with STALE processes after deciding whether evidence must be preserved.

B.8 Windows Server: identify the PHP that IIS invokes (Code 17)

The following PowerShell script reads the IIS configuration file (applicationHost.config) and outputs the executable to which the *.php handler passes requests, the PHP registered as a FastCGI application, the location of php.ini (the PHPRC environment variable), the displayed version and the SHA-256 hash. It only reads and changes no settings.

Code 17. PowerShell: record the actual PHP, its version and hash from the IIS configuration b8_iis_php.ps1

```powershell
# Identify the PHP that IIS actually invokes, from applicationHost.config (read-only)
param(
  [string]$Config = "$env:windir\System32\inetsrv\config\applicationHost.config"
)
[xml]$cfg = Get-Content -LiteralPath $Config -Raw

# (1) Which executable handles *.php (handler mappings)
foreach ($h in $cfg.SelectNodes("//handlers/add[@path='*.php']")) {
  'handler {0}: {1}' -f $h.name, $h.scriptProcessor
}

# (2) The FastCGI application itself. Even when the displayed version
#     is the same, the package can differ, so record SHA-256 too
foreach ($app in $cfg.SelectNodes('//system.webServer/fastCgi/application')) {
  $exe = $app.fullPath
  $rc  = $app.SelectSingleNode("environmentVariables/environmentVariable[@name='PHPRC']")
  'FullPath : {0}' -f $exe
  'PHPRC    : {0}' -f $(if ($rc) { $rc.value } else { '(not set)' })  # location of php.ini
  'Version  : {0}' -f (& $exe -v | Select-Object -First 1)
  'SHA256   : {0}' -f (Get-FileHash -Algorithm SHA256 -LiteralPath $exe).Hash
}
```

Output (PowerShell 7.4.6 reading a test configuration file; the leading part of paths is shortened to "…")

```
handler PHP_via_FastCGI: …/PHP/php-cgi.exe
FullPath : …/PHP/php-cgi.exe
PHPRC    : C:\PHP
Version  : PHP 8.3.6 (cli) (built: Oct 19 2026 15:16:58) (NTS)
SHA256   : 7F1AC2FE89D267364311651C0E7E6651D32EA006C9A701C0CAB68EB2394A67D7
```

The test was run with PowerShell 7.4.6 on Linux, reading a configuration file modelled on the Windows format and targeting the PHP CLI placed under the name php-cgi.exe; that is why the version shows "(cli)". No run tests were performed on Windows Server or IIS. Handlers can be overridden in a site's web.config, so check in IIS Manager too. Compare the SHA-256 with the value shown on the distributor's download page, because, as with the 26 October rebuilds, the contents can differ even when the displayed version is the same. [85]

Appendix C. Questions and answers for policymakers

Q1. If old languages and operating systems are dangerous, why not rewrite everything?

> Rewriting at once the operating systems and language runtimes on which public administrations worldwide depend is unrealistic in terms of cost, people, time and compatibility. Rewriting also risks creating new defects along the way. This paper proposes first lowering the probability and scale of harm through DevSecOps (stage one), then replacing, as a priority, the high-risk components that handle external input (stage two).

Q2. Is it not enough to block attacks from abroad?

> No. Attacks arrive through hijacked domestic devices, stolen legitimate accounts and tampered components. Judgements based on the source country are only supporting information. Each service needs authentication and authorisation, and a structure in which an intrusion does not spread (zero trust).

Q3. Are we safe if we install a WAF (web application firewall)?

> In this incident, WAFs were a useful layer of defence even before disclosure. But they have no effect on traffic that bypasses them, and mutated attacks were observed. A WAF is no substitute for the fix; it is one of the layers that buy time until the fix.

Q4. Does moving to the cloud, a VPS or containers solve the problem?

> Containers and serverless make compartmentalisation and replacement easier, but execution environments may be reused, and information stolen during the first request does not come back. If privileges are misconfigured, compartmentalisation does not work either. What matters is not the introduction of a mechanism but verified operations that include privileges, detection and recovery. The same applies to a VPS: virtualisation boundaries separate customers but do not protect against defects or secrets inside each server (Section 6).

Q5. Was it a mistake to shut services down?

> When reliable deployment of the fix and confirmation of whether a compromise occurred are not possible, a precautionary shutdown can be a rational decision. The problem was that there was no option other than shutdown. With an asset register and the capability for rapid deployment, the scope and duration of shutdowns can be greatly reduced.

Q6. Will rewriting code with AI make it cheaper, faster and safer?

> AI is a powerful aid, but successful compilation does not guarantee that meaning has been preserved. Require an accountable human owner, clear specifications, testing that includes malicious input, formal verification and independent review, and replace in small steps starting with high-risk components.

Q7. How should we explain the effect?

> Measure the metrics in Section 11 (public asset coverage, time to deploy fixes, privilege escalation blocking rate, recovery test success rate and so on) before and after adoption, and publish them. Show with evidence not that "we are absolutely safe", but that "we have reduced risk by this much and can recover this quickly".

Appendix D. Glossary

Term	Meaning
CVE / CVSS	The common identifier assigned to a vulnerability / a scoring method that expresses severity numerically. Scores can differ between assessors.
KEV	CISA's catalogue of vulnerabilities whose exploitation has actually been confirmed. Federal agencies are given remediation deadlines.
Zero-day	A vulnerability exploited before a fix is published, or such an attack.
Embargo	An agreement to keep vulnerability information among the parties concerned until a fix is ready. In this incident it was broken before disclosure.
PoC	Proof-of-concept code demonstrating that an attack is possible.
OOM / UAF / RCE	Out of memory / use after free (illegitimate use of freed memory) / remote code execution. The three are not synonyms; OOM does not always lead to UAF or RCE.
Memory safety	The property that access outside allocated bounds or to freed memory is prevented by the language or runtime.
SAPI / mod_php / PHP-FPM	The interface connecting PHP to a web server or other host / the mode embedded in Apache / the mode running as an independent pool of processes.
Worker	A process that handles requests. For performance, it is reused for many requests.
GC / FFI	A mechanism that automatically reclaims memory no longer needed / a mechanism for calling functions in other languages (which also imports the dangers of external libraries).
WAF	A device or service that inspects web traffic and blocks what appears to be an attack.
DevSecOps	A continuous way of working that links development, security and operations through the same change management and chain of evidence.

Term	Meaning
CI / CD	Continuous integration / continuous delivery (or deployment).
SBOM	Software bill of materials. A record of the components included; not a certificate of safety.
SAST / DAST	Static testing that analyses source code and the like / dynamic testing that examines a running application. Neither guarantees finding every defect.
Digest	A unique value computed from an artefact's content. The same value means the same content.
Zero trust	The approach of not treating network location as grounds for trust, and authenticating and authorising each request.
Least privilege	The principle of giving each component only the minimum privileges its function requires.
SIEM / SOAR	A monitoring platform that aggregates and analyses multiple logs / a mechanism that orchestrates and automates response procedures. The privileges and stop conditions of automation are also subject to management.
RTO / RPO	Recovery time objective / recovery point objective (the tolerated window of data loss). Their feasibility is measured through recovery drills.
Backport	Bringing a fix from a newer release into an older one without changing the version number.
VPS	Virtual private server. A hosting arrangement in which a physical server is partitioned by virtualisation and rented to each customer as if it were dedicated. Responsibility for updates inside the guest OS normally lies with the customer.

References

Sources accessed on 26 October 2026 (Japan time). Living documents may change. If a URL has moved or is restricted, locate the issuer using the title and identifier. References 1–49 come from the original draft; 50 onward were added in this revision.

[1] PHP Group (2026). Security advisory, 21 October 2026; Session upload progress notice, 23 October 2026. PHP News Archive.
https://www.php.net/archive/2026.php

[2] PHP / php-src. RCE via OOM corruption, GHSA-w8hw-w9fx-j79w (CVE-2026-109618); Susceptibility and Mitigation.
https://github.com/php/php-src/security/advisories/GHSA-w8hw-w9fx-j79w

[3] PHP / php-src. RCE via stale global state, GHSA-6wmf-4rv5-p3rp (CVE-2026-109619); Producers, Consumers.
https://github.com/php/php-src/security/advisories/GHSA-6wmf-4rv5-p3rp

[4] JPCERT/CC (2026). Alert on vulnerabilities in PHP, at260033 (in Japanese).
https://www.jpcert.or.jp/at/2026/at260033.html

[5] Amazon Threat Intelligence (2026). PHP2Shell exploitation identified prior to public disclosure in MadPot.
https://aws.amazon.com/blogs/security/amazon-threat-intelligence-identifies-php2shell-exploitation-prior-to-public-disclosure-in-madpot/

[6] Ministry of Finance, Japan (21 October 2026). Notice of temporary suspension in response to the PHP vulnerabilities (in Japanese).
https://www.mof.go.jp/public_relations/ohter/20261021231537.html

[7] Ministry of Finance, Japan (22 October 2026). Notice of resumption of publication (in Japanese).
https://www.mof.go.jp/public_relations/ohter/20261022184327.html

[8] CERT-EU (2026). Critical Vulnerabilities in PHP, Security Advisory 2026-018.
https://cert.europa.eu/publications/security-advisories/2026-018/

[9] GNU C Library (17 April 2024). GLIBC-SA-2024-0004 / CVE-2024-2961: iconv ISO-2022-CN-EXT buffer overflow.
https://www.openwall.com/lists/oss-security/2024/04/17/9

[10] J. Postel (1981). RFC 801: NCP/TCP Transition Plan.
https://www.rfc-editor.org/info/rfc801

[11] R. M. Metcalfe (1973). RFC 602: The Stockings Were Hung by the Chimney with Care.
https://www.rfc-editor.org/rfc/rfc602.html

[12] D. M. Ritchie. The Evolution of the Unix Time-sharing System.
https://www.nokia.com/bell-labs/about/dennis-m-ritchie/hist.html

[13] The Open Group. History and Timeline of UNIX (1969, 1989, 1993 entries).
https://unix.org/unix_history.html

[14] FreeBSD Project. BSD family tree.
https://cgit.freebsd.org/src/tree/share/misc/bsd-family-tree

[15] FreeBSD Project. Explaining BSD.
https://docs.freebsd.org/en/articles/explaining-bsd/index.html

[16] OpenSSH Project. Project History.
https://www.openssh.org/history.html

[17] IBM Japan. Corporate history (2000: investment of $1 billion in Linux).
https://www.ibm.com/jp-ja/about/ibm-japan-history

[18] Carnegie Mellon University. Overview of the Mach Project.

https://www.cs.cmu.edu/afs/cs.cmu.edu/project/mach/public/www/overview.html

[19] D. Golub, R. Dean, A. Forin, R. Rashid (1990). Unix as an Application Program. USENIX Summer Conference.
https://www.cs.cmu.edu/afs/cs/project/mach/public/www/doc/abstracts/mach3_intro.html

[20] Apple. XNU source repository, README.
https://github.com/apple-oss-distributions/xnu

[21] PHP Group. History of PHP.
https://www.php.net/manual/en/history.php.php

[22] PHP Group. Description of core php.ini directives.
https://www.php.net/manual/en/ini.core.php

[23] LispWorks. Common Lisp HyperSpec: UNWIND-PROTECT.
https://www.lispworks.com/documentation/HyperSpec/Body/s_unwind.htm

[24] LispWorks. Common Lisp HyperSpec: HANDLER-BIND.
https://www.lispworks.com/documentation/HyperSpec/Body/m_handle.htm

[25] LispWorks. Common Lisp HyperSpec: RESTART-CASE.
https://www.lispworks.com/documentation/HyperSpec/Body/m_rst_ca.htm

[26] Erlang/OTP. Robustness.
https://www.erlang.org/doc/system/robustness.html

[27] Erlang/OTP. erl_nif: “A native function that crashes will crash the whole VM.”
https://www.erlang.org/doc/apps/erts/erl_nif.html

[28] Rust Project. The Rustonomicon: What Unsafe Rust Can Do.
https://doc.rust-lang.org/nomicon/what-unsafe-does.html

[29] Rust Project. The Rust Programming Language: Unrecoverable Errors with panic!.
https://doc.rust-lang.org/book/ch09-01-unrecoverable-errors-with-panic.html

[30] K. Havelund, M. Lowry, S. Park, C. Pecheur, J. Penix, W. Visser, J. White (2000). Formal Analysis of the Remote Agent Before and After Flight. NASA.
https://ntrs.nasa.gov/citations/20000055731

[31] NASA/JPL (7 June 1999). Remote Agent Experiment Meets All Objectives.
https://www.jpl.nasa.gov/news/remote-agent-experiment-meets-all-objectives

[32] AWS. Understanding the Lambda execution environment lifecycle.
https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtime-environment.html

[33] Firecracker Project. README.
https://github.com/firecracker-microvm/firecracker

[34] Kubernetes Project. Security.
https://kubernetes.io/docs/concepts/security/

[35] Kubernetes Project. Pod Security Standards.
https://kubernetes.io/docs/concepts/security/pod-security-standards/

[36] Kubernetes Project. Network Policies.
https://kubernetes.io/docs/concepts/services-networking/network-policies/

[37] NIST (2022). SP 800-218: Secure Software Development Framework (SSDF), Version 1.1.
https://csrc.nist.gov/pubs/sp/800/218/final

[38] NIST (2020). SP 800-207: Zero Trust Architecture.
https://csrc.nist.gov/pubs/sp/800/207/final

[39] CISA, NSA, FBI, ASD's ACSC, CCCS, NCSC-UK, NCSC-NZ, CERT NZ (December 2023). The Case for Memory Safe Roadmaps.
https://www.cisa.gov/resources-tools/resources/case-memory-safe-roadmaps

[40] GitHub. Secure use reference for GitHub Actions.
https://docs.github.com/en/actions/reference/security/secure-use

[41] GitHub. About protected branches.
https://docs.github.com/en/repositories/configuring-branches-and-merges-in-your-repository/managing-protected-branches/about-protected-branches

[42] Trivy. Filtering.
https://trivy.dev/docs/latest/configuration/filtering/

[43] Falco. Rules.
https://falco.org/docs/concepts/rules/

[44] PeachPie. Documentation.
https://docs.peachpie.io/

[45] ISO/IEC JTC1/SC22/WG14 (2011). N1570 (C11 committee draft), 7.13.2.1 The longjmp function.
https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf

[46] CMUCL. User's Manual: Credits.
https://cmucl.org/docs/cmu-user/html/Credits.html

[47] NGINX. Module ngx_http_core_module: client_max_body_size, client_body_buffer_size.
https://nginx.org/en/docs/http/ngx_http_core_module.html

[48] Cognitive Research Labs, Inc. Government projects: government projects of the Cognitive Research Laboratories era (in Japanese).
https://crl.co.jp/government-projects

[49] Cognitive Research Labs, Inc. Government Projects.
https://crl.co.jp/en/government-projects

[50] A. Le Blanc, I. Tovilo, J. Zelenka, N. Dossche, N. Grekas (18 October 2026). Avoid state corruption via bailout. php-src commit 31c0ea7 (PHP-8.5 branch).
https://github.com/php/php-src/commit/31c0ea7d7ff1d1fccb6492e52dab544011628065

[51] Ubuntu (19 October 2026). php8.3 8.3.6-0ubuntu0.24.04.13 (noble-security) changelog: CVE-2026-109618_109619.patch.
https://launchpad.net/ubuntu/+source/php8.3

[52] piyokango (2026). A summary of the two critical PHP vulnerabilities (CVE-2026-109618 and CVE-2026-109619). piyolog, updated through 24 October 2026 (in Japanese).
https://piyolog.hatenadiary.jp/entry/2026/10/21/174516

[53] Palo Alto Networks Unit 42 (2026). Threat Brief: Critical Vulnerabilities in PHP: CVE-2026-109618, CVE-2026-109619 (updated 25 October 2026; the 23 October addition on a European government entity was withdrawn).
https://unit42.paloaltonetworks.com/cve-2026-109618-cve-2026-109619/

[54] Canadian Centre for Cyber Security (21 October 2026, updated 23 October). AL26-026: Critical vulnerabilities impacting PHP.
https://cyber.gc.ca/en/alerts-advisories/al26-026-critical-vulnerabilities-impacting-php-cve-2026-109618-cve-2026-109619

[55] Recorded Future Insikt Group (21 October 2026). PHP2Shell Exploitation: Two Critical PHP Flaws Explained.
https://www.recordedfuture.com/blog/php2shell-vulnerability-exploitation

[56] Jiji Press / Nippon.com (21 October 2026). Japan PM Warns of Vulnerabilities in PHP Programming Language.
https://www.nippon.com/en/news/yjj2026102101249/

[57] AAP (22 October 2026). Government websites go dark in software flaw scramble.
https://www.portnews.com.au/story/9373041/government-websites-go-dark-in-software-flaw-scramble/

[58] The Canadian Press / CTV News (21 October 2026). Some government websites taken down in response to security risk: AI minister.
https://www.ctvnews.ca/sci-tech/article/some-government-websites-taken-down-in-response-to-security-risk-ai-minister/

[59] The Hacker News (October 2026). PHP2Shell Exploitation Attempts Continue Amid Suspected N. Korea Global Scanning Activity.
https://thehackernews.com/2026/10/php2shell-exploitation-hits-government.html

[60] PPC Land (2026). Cloudflare blocks php2shell exploits 9 hours before PHP flaws went public.
https://ppc.land/cloudflare-blocks-php2shell-exploits-9-hours-before-php-flaws-went-public/

[61] Kredo (24 October 2026). PHP vulnerability news, October 2026: cases in Japan, government statements and timeline (in Japanese).
https://kredo.jp/media/php-vulnerability-news-2026-october/

[62] Bitsight (21 October 2026). CVE-2026-109618 and CVE-2026-109619 (PHP2Shell): Two Critical PHP Remote Code Execution Flaws.
https://www.bitsight.com/blog/critical-vulnerability-alert-cve-2026-109618-and-cve-2026-109619-php2shell

[63] Norwegian Defence Research Establishment (FFI). From the US to FFI: Kjeller was first online (15 June 1973).
https://www.ffi.no/en/news/from-the-us-to-ffi-kjeller-was-first-online

[64] E. H. Spafford (1988). The Internet Worm Program: An Analysis. Purdue Technical Report CSD-TR-823.
https://www.cerias.purdue.edu/apps/reports_and_papers/search/index/1515

[65] The New Stack. NASA Programmer Remembers Debugging Lisp in Deep Space.
https://thenewstack.io/nasa-programmer-remembers-debugging-lisp-in-deep-space/

[66] R. Garret (2002). Lisping at JPL.
https://flownet.com/gat/jpl-lisp.html

[67] D. M. Ritchie (1993). The Development of the C Language.
https://www.nokia.com/bell-labs/about/dennis-m-ritchie/chist.html

[68] NSA (November 2022; updated April 2023). Software Memory Safety, Cybersecurity Information Sheet.
https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF

[69] NSA and CISA (June 2025). Memory Safe Languages: Reducing Vulnerabilities in Modern Software Development (press release).
https://www.nsa.gov/Press-Room/Press-Releases-Statements/Press-Release-View/Article/4223298/

[70] The White House ONCD (26 February 2024). Back to the Building Blocks: A Path Toward Secure and Measurable Software.
https://bidenwhitehouse.archives.gov/oncd/briefing-room/2024/02/26/memory-safety-fact-sheet

[71] CISA and FBI (January 2025, version 2). Product Security Bad Practices.
https://www.cisa.gov/resources-tools/resources/product-security-bad-practices

[72] CISA (3 November 2021). Binding Operational Directive 22-01: Reducing the Significant Risk of Known Exploited Vulnerabilities.
https://www.cisa.gov/binding-operational-directive-22-01

[73] European Union (2024). Regulation (EU) 2024/2847 (Cyber Resilience Act), Articles 14 and 71.
https://eur-lex.europa.eu/eli/reg/2024/2847/oj/eng

[74] Cabinet Office, Japan (2025). On the enforcement of the Act on the Prevention of Damage Caused by Unauthorized Acts against Important Electronic Computers (Act No. 42 of 2025) (in Japanese).
https://www.cao.go.jp/cybersecurity/pdf/sekousetsumei.pdf

[75] W3Techs (accessed 25 October 2026). Usage statistics of PHP for websites.
https://w3techs.com/technologies/details/pl-php

[76] DORA / Google Cloud (2021). Accelerate State of DevOps 2021.
https://gitlab.com/gitlab-com/www-gitlab-com/uploads/069ee8e2ee6af463cf0aafcd89eda33e/state-of-devops-2021.pdf

[77] GCC. Common Variable Attributes: cleanup.
https://gcc.gnu.org/onlinedocs/gcc/Common-Variable-Attributes.html

[78] LispWorks. Common Lisp HyperSpec: WITH-OPEN-FILE.
https://www.lispworks.com/documentation/HyperSpec/Body/m_w_open.htm

[79] M. K. McKusick (1999). Twenty Years of Berkeley Unix. In Open Sources, O'Reilly.
https://www.oreilly.com/openbook/opensources/book/kirkmck.html

[80] NIST (December 2025). SP 800-218 Rev. 1 (SSDF Version 1.2), Initial Public Draft.
https://csrc.nist.gov/pubs/sp/800/218/r1/ipd

[81] INTERNET Watch (21 October 2026). FX automated-trading VPS "Onamae.com Desktop Cloud" suspends servers after unauthorised access (in Japanese).
https://internet.watch.impress.co.jp/docs/news/2149829.html

[82] INTERNET Watch (26 October 2026). GMO issues a detailed report on the cyberattack on "Onamae.com Desktop Cloud" and an urgent request for "active cyber defence" (in Japanese).
https://internet.watch.impress.co.jp/docs/news/2150991.html

[83] piyokango (23 October 2026, updated 24 October). A summary of the unauthorised access to Onamae.com Desktop Cloud. piyolog (in Japanese).
https://piyolog.hatenadiary.jp/entry/2026/10/23/014520

[84] PHP Foundation Discourse, php.announce (26 October 2026). [ANNOUNCE] Updated PHP Windows builds for Brotli security fix.
https://discourse.thephp.foundation/t/announce-updated-php-windows-builds-for-brotli-security-fix/6324

[85] The Hacker News (26 October 2026). PHP Issues Second Windows Security Update in Five Days to Fix Brotli Code Execution Flaw.
https://thehackernews.com/2026/10/php-issues-second-windows-security.html

[86] Microsoft Learn. IIS configuration reference: Adding Handlers <add>.
https://learn.microsoft.com/en-us/iis/configuration/system.webServer/handlers/add

[87] AWS Well-Architected Framework, Security Pillar. Shared responsibility.
https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/shared-responsibility.html

[88] The FreeBSD Project. FreeBSD Handbook, Chapter 17: Jails and Containers.
https://docs.freebsd.org/en/books/handbook/jails/

[89] GMO Internet Group (24 October 2026, 11:50). On the cyberattack on the Onamae.com Desktop Cloud service (4th report) (in Japanese).
https://group.gmo/info/alert/20261024/

[90] GMO Internet Group (26 October 2026, 23:10). Current response to the unauthorised access to Onamae.com Desktop Cloud (7th report) (in Japanese).
https://group.gmo/info/alert/2026102602/