Security Now! #1102 - 10-27-26

A Strategic Plug Pull

This week on Security Now!

• We take a detailed look at the many facets and aspects of last week's wild PHP ride:
• What happened. • How it happened. • Why it happened. • How it could have been prevented.
• How it impacted many of our listeners. • We consider the implications of what is said to be coming next and the idea of temporary "strategic disconnection" from the Internet.

It's not a bug… it's an inherent artifact of LLM technology

Technical Debt Repayment

This week, we're going to break from this podcast's longstanding tradition of eventually getting around to its main topic because there's nothing more I want to talk about than what happened last week with PHP. There are so many interesting aspects and sides to the story that surrounds last week's truly earth shaking events.

The first good news is that "earth shaking" much more accurately describes what happened than "earth shattering". The earth was not shattered, we're all still here; but it was certainly given a good shaking. Before we get into an after-the-event debrief, let's catch everyone up on the timeline surrounding what happened:

The bugs that enabled last week's pre-auth remote code execution were first introduced in PHP's version 5.4 release on March 1st, 2012, and every release of PHP since then, until last week's multiple fixes to the various v8 branches have been vulnerable. PHP has suffered from perhaps more than its share of serious security vulnerabilities through the years. At this point in my recounting of PHP's past troubles, I was initially planning to provide specifics to back up my general statement of PHP's trouble security history. But as I started enumerating them, the list and their details became so long that I realized there was no point. Besides, everyone, certainly those listening to this podcast for the past 20+ years, already knows this to be true.

One tidbit I will share from the list of previous PHP nightmares was to recall how many problems PHP has suffered from its "unserialize" function. We've talked too often about the problems with deserialization. The idea is that a language can be asked to save a complex object which is typically represented in memory in a data structure. To save that in-memory data structure for storage, the object must be converted into a byte-stream using an object-aware process we call serialization. That's the easy half of the problem. It's reversing the process where it's so easy to fall down; and most implementations do for a while. Turning a previously serialized object back into the object it represents requires an interpreter to read the byte stream, and doing this safely in the face of deliberate corruptive intent turns out to be surprisingly difficult. PHP has more trouble in this regard than probably any other language system.

PHP's well-deserved reputation for security vulnerabilities created a dilemma for me when I wanted to create web forums for GRC. Up to that point, I had hand-written every bit of application software running on GRC's server infrastructure, all in assembly language, of course. But all of the forum software I was considering sat on top of PHP. As I mentioned on the podcast at the time, and many times since, there was no way I was willing to have a publicly exposed machine running PHP within GRC's server infrastructure. My solution was to spin up an entirely separate server and sequester it from the rest of the network behind a dedicated hardware firewall. Doing that would at least limit any damage a PHP-facilitated compromise of that server might cause. Since then, I've added GRC's PHP-based NeuvoMailer for our email backend, and the grc.sc link shortener. For much of the web, even today in 2026, PHP is a necessary and painful reality upon which a huge number of systems depend. Here, I will take a moment to back that up for the benefit of those who may not realize just how prevalent PHP has become:

To start with, PHP has always been a popular language for building web content management systems. It's the language used by WordPress, Drupal, Joomla, TYPO3, Craft CMS, Concrete

CMS, and Grav. It is used for eCommerce by WooCommerce, Magento / Adobe Commerce, PrestaShop, OpenCart and Shopware. The two popular Wikis: MediaWiki, which runs Wikipedia and most other wikis and DokuWiki were both authored in PHP. I mentioned that web forums were nearly all written on top of PHP. That includes XenForo, the forum system I carefully chose and have never regretted. And also phpBB, vBulletin, Invision Community and Flarum. In education, Moodle is the dominant open-source learning management system. Both Nextcloud and its predecessor ownCloud are PHP-based. The classic MySQL/MariaDB administration interface is phpMyAdmin and both cPanel and Plesk were implemented in PHP. PHP has been a favorite CRM implementation language hosting SuiteCRM and SugarCRM, Mautic for marketing automation and Dolibarr for ERP and CRM. The large PHP-based web frameworks include Laravel, Symfony, CodeIgniter, CakePHP, Yii and Laminas which was originally the Zend Framework. And finally, countless less universal systems and apps require PHP, like NuevoMailer which I chose. It's pretty much everywhere.

Leveraging the new power of AI for vulnerability discovery and exploit generation that emerged this year, Google's Wiz Security and Calif Security independently discovered and privately reported the critical vulnerabilities that caused such havoc last week. Both groups knew that too much of the Internet was using PHP to make light of a vulnerability that has been present in the past 14 years of PHP when used in its default configuration.

Last Wednesday, OpenAI — whose GPT AI was used to make this recent discovery using their "Daybreak" and "Patch the Planet" programs — posted about all this under the headline: *"Patch the Planet: global response coordination as agents uncover critical vulnerabilities"*. It's naturally somewhat self-congratulatory, but it contains important information which will be useful for navigating the next several years of AI-driven vulnerability discovery and patching. They wrote:

> *Over the past two months, we led a coalition of maintainers, researchers, and security providers to address critical vulnerabilities in PHP, an open-source programming language used to run websites and web applications. Calif identified a critical flaw using GPT-5.6 Sol as part of our Patch the Planet initiative, with Wiz also independently discovered and reported both vulnerabilities. PHP's maintainers led the fixes and release, with our teams and partners like Wiz, Cloudflare, and others contributing research, technical support, and coordination to prepare protections ahead of disclosure.*
>
> *For organizations using PHP, the priority now is applying the fixes. We're sharing a PHP version checker as a Codex skill to help operators identify potentially vulnerable PHP versions and, alongside PHP's and their providers' guidance, determine which updates apply.*
>
> *Given the severity of these flaws and PHP's widespread use, we consider this one of the most impactful security issues in recent memory. It highlights how AI can uncover security flaws with internet-scale impact, but the same capabilities can also help attackers rapidly develop offensive tools and exploits.*
>
> *To our knowledge, frontier AI found these vulnerabilities before attackers did, opening a Defender's Window for maintainers and security providers to prepare fixes and protections ahead of disclosure.*

To help more defenders use these capabilities, we've committed $1 billion in subsidized Daybreak access through Daybreak for Frontline Defenders, focused on organizations with limited security resources protecting essential services.

As significant as they are, these bugs are among several significant vulnerabilities we've already found and are working to fix, and we expect similar findings to increase sharply in the coming months.

I want to highlight that. We're going to hear something similar a bit later from CloudFlare's Matthew Prince. What becomes evident is that this PHP secret, which was closely held for two months before it leaked two days before its planned release date, is just one of the apparently critical discoveries that have been made and are currently embargoed as fixes are being prepared and coordinated. That makes what follows all the more important. OpenAI continues:

Drawing on our experience thus far, we're sharing a proposed approach, appropriate for the AI era, for responding to critical vulnerabilities in widely used software, and we're asking the security community to help improve this framework. It brings together our models, security expertise, and maintainers to help turn findings into protection and build a response that can scale as AI capabilities advance. Getting a fix into use can mean adapting it for older releases, preparing provider mitigations, and updating systems across many organizations. A repeatable process helps those teams act quickly as more vulnerabilities are discovered.

Through Patch the Planet, we contributed security expertise and model access to support the PHP response, and worked with maintainers and research partners to test fixes and prepare guidance. We also helped coordinate with infrastructure providers on protections for affected systems and engaged government cyber agencies to help guidance reach critical organizations. This supported the maintainer-led response and helped other defenders prepare.

For some findings, researchers can work directly with the software's maintainers to investigate the issue and develop a fix. Others need a wider response because they affect many products or because distributors and service providers need time (which may be in short supply if frontier capabilities are not too far ahead of those that are broadly available) to prepare updates and protections. In those cases, the proposed framework can help maintainers identify who needs to be involved and coordinate the work. The response can involve five stages:

1. *Validate and fix the upstream vulnerability. The maintainer, researchers, and essential technical partners confirm the finding, assess its impact, and develop and test a fix. This process includes adversarial testing with frontier models to look for ways around the patch and related variants of the vulnerability.*

2. *Give affected organizations time to act. We propose a framework for considering which organizations need advance information to prepare protections, while maintainers decide whom to involve. We help with outreach, information-sharing agreements, and coordination between teams. A distributor might adapt a fix for an older release, a hosting provider might test a mitigation, and a security provider might prepare detection or prevention measures, including web application firewall (WAF) rules.*

3. *Coordinate with government partners to help protect critical systems. Government agencies play an essential role in coordinating a wider response and helping security guidance reach organizations responsible for critical infrastructure and other essential services. In the case of PHP, we worked, in collaboration with our partners, to engage with multiple parts of the U.S. government, including the White House's Super Intelligence Force, the Office of the National Cyber Director (ONCD), the Cybersecurity and Infrastructure Security Agency (CISA), as well as international partners to support that effort.*

4. *Prepare updates across affected products. As the fix, guidance, and disclosure plans become stable, maintainers can involve additional distributors, hosting and service providers, response teams, and operators to prepare updates and address gaps in coverage.*

5. *Once a patch is available, publish guidance and support deployment. Public guidance helps people assess their exposure, find authoritative updates or interim mitigations, and check that a fix is running. The response continues as operators deploy updates and report where they need help.*

The proposed framework considers an organization's protective role, including protecting its own systems and customers, the time needed to act, and its ability to handle sensitive information. The same criteria would apply to comparable organizations, with maintainers deciding whom to involve ahead of disclosure.

Each participant receives the information needed to carry out their protective work, from network logs with indicators of compromise to more sensitive working proof-of-concept exploits. Even limited details can reveal a vulnerability, so we weigh the risk of sharing them against the protection they enable.

Teams following this framework would work with maintainers to give defenders time to prepare and reassess the timeline if there's active exploitation in the wild, a leak, a revealing public patch, or increased risk to users from stalled remediation.

The Patch the Planet website brings together public disclosures and guidance to help operators identify affected systems, find the right update, and verify that the updated software is running properly. For PHP, it points to the maintainers' official patch and upgrade instructions and our partners' technical research. The guidance also covers interim mitigations, when to involve a provider or expert, and backup and recovery. Interim protections can reduce risk, but we'll make their limits and operational tradeoffs clear.

Their posting goes on from there, becoming more generic and a bit more OpenAI-centric. I'm unsure what role any given AI provider should really have in this. They are providing a tool that was used to facilitate this discovery. It seems to me that the discoverer of a massively impactful flaw needs to very quietly and carefully inform the publishers or maintainers of the system in question and then those publishers or maintainers are best positioned to best know how to proceed, since every such software system will have wildly differing demographics, usage, update patching and exploit consequences.

It does appear clear, however, that last week's frantic PHP event is already known to only be the first of at least several more similar events that will be arriving before long.

Okay, so what happened last week? The guys at VulnCheck checked in last Thursday about this, writing:

> *Earlier this week, PHP disclosed two critical vulnerabilities, CVE-2026-109618 and CVE-2026-109619 ("PHP2Shell"), that allow for unauthenticated remote code execution. VulnCheck researchers independently reproduced both vulnerabilities the night the news leaked. What have we seen?*
>
> *VulnCheck's own exploit achieves RCE on ASLR-enabled systems: Our Initial Access Intelligence team has a working RCE exploit for CVE-2026-109618 on x86-64 with ASLR enabled on several PHP 8.4.x versions.*
>
> *We see massive exposure: VulnCheck Target Intelligence found PHP on more than 2.6 million internet-facing hosts, and 99.998% of them are vulnerable or likely vulnerable. There is no fix for many: About 1.1 million (42%) are running end-of-life PHP branches with no official patched release. And PHP is often hidden: More than 469,000 WordPress and more than 124,000 Laravel hosts run on PHP. Even if a system doesn't advertise PHP, it can still be exposed.*
>
> *Exploitation is now active in the wild: VulnCheck Canaries have recorded growing numbers of attempted PHP2Shell exploitation. We have observed attackers adopting at least five public exploit kits within a day, including one multi-stage attempt to bypass ASLR.*
>
> *There is also a flood of bad PoCs: Our research team has reviewed hundreds of public proof-of-concept repositories, most of which are fake or otherwise demonstrate artificial lab scenarios. VulnCheck Exploit and Vulnerability Intelligence data is being updated continuously as our team reviews new PoC code.*

Okay. So we are seeing the familiar patching pattern play out once again, but this time on a far more sizeable and dramatic scale. It is literally the case that AI enabled the discovery of a flaw that allowed most of the Internet's assets to be remotely compromised. That's huge.

Before we look at Matthew Prince's comments it's worth appreciating that this flaw had been known of for many years by the responsible people but had been deliberately ignored because its exploitation had not been demonstrated. Think about that for a second and ask yourself where else similar known but dormant problems lie. Here's a piece of the timeline description posted by a Calif author, who writes:

> *The year was 2004. The iPhone was yet to be invented and our very own Stefan Esser was still the PHP security guy. Back then, I was a kid. I've followed Stefan's PHP research to improve my own understanding of PHP as an attack surface. Long before the 2008 call-time-pass-by-reference / interruption vulnerabilities and the more famous unserialize() use-after-frees and POP(object) injections, Stefan unearthed the first PHP memory limit remote vulnerability in 2004. I remembered the vulnerability as it was described by Stefan as being critical enough that he would not release his exploit to the public. Eventually, two other commercial exploit vendors developed non-public exploits as well - CORE and Immunity Inc. Other than those, we only had fake exploits circulating on IRC designed to compromise your machine.*

The years went on and all of the default unserializable classes with use-after-free or type confusion bugs in PHP were found and patched as well. We verified this earlier in the year when we released our first PHP research attempting to find any remnants that might have been missed; we found no exploitable vectors that would be exploitable by default.

Then, in 2020 came a report of a seemingly harmless bug #78876 to the PHP project, noting that certain long variables could cause an out-of-memory condition and the failure to clean up temporary files. At the time, not much was made of the report. Fast forward to 2024 when OSTIF and Quarkslab performed an audit of PHP and rediscovered the same OOM issue. However, exploitation was not proven, hence the bug was not fixed.

Finally, we arrive in 2026. OpenAI invited us to join its Patch the Planet initiative, and PHP stood out as a high-impact target. We wanted the PHP Foundation involved, so Stefan connected us with Volker Dusch, who accepted our invitation. Then we got to work hacking PHP.

On August 1st, 2026, we made the discovery. We were not expecting to find anything critical - everyone and every lab had already done extensive AI assisted audits of this target. But, only a few days after starting the audit and asking the right questions related to Stefan's earlier findings, we managed to surface the first OOM-fatal flaw and the first of the three interruption points that allowed us to piece together an RCE against WordPress.

Since the first bug isn't usually the best one, we continued looking for better variants. In the end, we found two more that lead to reliable exploits against mod_php and PHP-FPM.

What this means is that PHP is such a mess that even during a full security audit, fundamental architectural problems that do not directly and provably compromise PHP's security are ignored. Had these well known and long tolerated out-of-memory mishandlings been addressed years ago, none of what happened last week would have transpired. As I noted, it's going to be interesting to see what the next surprises reveal and whether we see a similar pattern.

To that end, I want to next share a piece that appeared in The Atlantic last Friday October 23rd. The piece, written by Matteo Wong who covers this area had the title *"Welcome to the Valley of Pain"* with the sub-head: *"An explosion of AI-powered hacks will reshape life online."* That's not the sub-heading I would give this piece since while I expect online life to be reshaped by AI — that's already underway — I doubt it will be directly reshaped by AI-aided malicious action. As I noted last week, behavioral change is more difficult but it's probably what will be needed.

So I believe we're going to go through and survive a rough patch, and that appears to be the feeling of the people Matteo's article quoted last week. He wrote:

This week could go down as the worst in the history of cybersecurity. On Wednesday, news emerged about two major vulnerabilities in PHP, a programming language used by 70 percent of websites, which prompted governments, hospitals, and companies around the world to take digital services offline as a precaution. There have already been millions of attempts to exploit these weaknesses.

But the PHP vulnerabilities were just the most attention-grabbing in a series of alarming revelations. Security flaws in, or successful hacks of, several major software products and platforms—including some used by Netflix, PayPal, Walmart, and other big names—were announced that same day. Yesterday, Quest Diagnostics, which runs lab tests for a third of American adults each year, disclosed that it, too, had been breached. When I asked Sam Rubin, a senior vice president at the cybersecurity firm Palo Alto Networks [a sponsor of the TWiT Network] *about the potential fallout from yet another hack—this one of the cloud platform Railway, which is used by Intuit and MGM Resorts, among others—he paused. "I didn't see that one," Rubin said. "It's been quite a busy week."*

Yet what's most alarming about the past few days is that by next year, this stretch of incidents—which some security experts are calling "Black October"—may be remembered as normal, or even tame. Matthew Prince, the CEO of the internet infrastructure and security company Cloudflare, told me: "It's going to feel like, over and over and over again, we're finding more and more bugs in very critical pieces of software. This story, which feels unique and fresh right now—in a few months, it's going to feel like we get one of these every week."

Prince said he knows of still more vulnerabilities that will be announced in the near future.

For the next year or two, if not longer, the internet is going to feel a bit touch and go, and more dangerous than we've come to expect. Matthew Prince said: "It's going to feel like the sky is falling for a while."

Okay. And then Cloudflare's Matthew Prince echoes the primary thesis I've been voicing for the past few months, and the reason for this topic's title "Technical Debt Repayment":

There's a decades-long backlog of bugs in software written without much review or scrutiny-which was more or less fine so long as there was a very small number of skilled hackers. Over the past two decades, companies, governments, and individuals have been lulled into a false sense of security: Most people don't think twice about punching credit-card numbers into the website of a local restaurant or gift shop, or forking over a home address and phone number in just about any customer survey. But the internet was always a dangerous place. We're just waking up from a trance.

AI agents can now be deployed as a cheap and near-infinite hacker army. In just the past couple of months, Rubin said, open-source models—which can be downloaded for free and used without restrictions—have become very good at finding vulnerabilities and automating attacks. In turn, the amount of malicious cyber activity has spiked. More websites will be defaced, more personal information will be stolen, and ransomware will proliferate. Rubin told me that the internet is entering a "valley of pain." When I posed the concern that this is the new normal to Adam Meyers, a senior vice president at the cybersecurity company CrowdStrike, he told me, "I think it's going to get a lot worse, actually."

Consider, for instance, another set of vulnerabilities disclosed this week, to the program Node.js. This software, which executes JavaScript code for dynamic webpages, is widely used by Fortune 500 companies, as well as in programs developed with AI. Meyers said: "These are the things the web runs on." Rubin said that because of how the platform is used, compromising just one software product that uses Node.js could infect tens of thousands of other organizations in what is known as a "supply-chain attack."

And the skill threshold for launching cyberattacks has become almost absurdly low: CrowdStrike tied a spate of cyberattacks on South Korean banks late last month to a 26-year-old in China. Meyers said: "After he got access to all the data, he turned around and asked AI, 'How do I monetize this? Where can I sell this?'. He didn't know what he was doing." Meyers said CrowdStrike is detecting "a new agentic adversary"—that is, an AI-powered hacking operation—"every two hours."

This is what we've talked about before. The effect AI has of lowering of the skill level required bar to the floor means that an entire new community of what were once wannabe hackers and attackers have become empowered overnight. The Atlantic's article continues:

Navigating this will require rapid and tremendous coordination from basically every sector. For instance, the vulnerabilities in PHP were discovered months ago by a consortium of AI and cybersecurity firms. Then, Prince said: "very quietly, for the last several months, we and other industry partners worked to effectively put in place protections. Basically, temporary shields until the weaknesses could be fully addressed. But "not everybody uses CloudFlare, and not everybody is quick to patch their software."

Some of the cybersecurity experts I've spoken with are optimistic about AI: Eventually, they say, AI will be used to find and address many of the major cybersecurity flaws, and the internet will become much safer overall. But before then, we may be stuck in a terrible limbo. Alex Stamos, the chief information-security officer at the AI company Cognition and a former chief security officer at Facebook told me: "This is going to last as long as it takes to rewrite and secure the billions of lines of code that have always had these vulnerabilities."

And even as cybersecurity experts rush to leverage AI to save the internet, the technology is introducing plenty of new problems. All vibe-coded apps, for starters, are very likely to contain security flaws. And at a time when people need to be more vigilant about their digital hygiene, multiple cybersecurity experts brought up the widespread adoption of AI assistants with more than a hint of scorn; as Rubin put it: "Our peers are using Muse and giving it carte blanche to use your credit card."

Just for the record, I'm going to push back on that a bit. We know that it's certainly possible for vibe-coding to produce insecure software. Many months ago we shared the story of a listener whose retired father asked AI to create a website for him. And the AI did what he asked. It created a website. Was it secure? Did it have any security enforcement? No. Because this guy's father hadn't known to, or thought to, ask for a "secure website" — he just said "website".

That's an extreme example, but it shows the trouble. There's no reason to believe that properly written software whose security is thoroughly tested and pounded on before its release will be any less secure than traditional software written without automation. And I suspect that future software will incorporate far more security. For one thing, we have much newer languages such as Rust which inherently resolve many of the problems that result from mistakes with earlier dangerous languages like C. Human coders continue choosing C because it's what they know. But an AI, unlike a human coder, can simply be told to please use Rust. So I believe that in the longer term there's reason for optimism on the software security front. But that's not today.

The Atlantic's article begins to wrap up, writing:

The internet is becoming a wilder place, and the path to secure it may require, in essence, a sort of pruning. Dan Guido, the CEO of the cybersecurity firm Trail of Bits, told me: "You'll likely see more companies take services off the internet when they don't need to be there," depreciating old webpages and forms so they cannot be targeted.

Individuals might be encouraged to delete unused browser extensions, he added, so that there are "fewer pieces of software that can access your data." Alex Stamos said: "People in general need to be more discerning of the companies they work with. You're just not gonna be able to assume every site you use is secure." That Pokémon trading-card website, fantasy- sports platform, or birding forum you use could be hacked for a password you've reused elsewhere, or a credit-card number, or worse. Other advice I heard: Don't host your own email client instead of Gmail. Don't host your own file system at home instead of using the cloud. Google and Dropbox have the resources to defend against an onslaught of AI hacks, and you don't.

Hold on a second. I would temper that a bit to say: Definitely do not host your own email client instead of gmail or your own file system if you do not fully understand the security implications of doing so." I assume when they talk about hosting your own file system they mean running a NAS. And we certainly know the danger of the temptation to have any management surface publicly exposed.

I'm also somewhat annoyed by the idea that advice is being given to just turn everything over to cloud-based services. That's certainly not appropriate for much of this podcast's audience, but, sadly, I can see that it might generic sense for the sake of security. The article concludes:

If, eventually, we do find and shore up all of the cybersecurity vulnerabilities out there, the result could be a more open web, where anybody can use widely available AI tools to run security checks at any time and you'll feel safer sharing your personal information again. But another possibility is an even more centralized internet, where only services and websites developed by large software providers are deemed trustworthy and reliable. The internet needs new walls to defend against AI-powered attacks. But in turn, we might also feel walled in.

I would hate to have the Internet consolidated into a handful of mega-service providers. But look at Cloudflare. I am more and more often encountering their "Verifying you're not a bot" interception page before I'm allowed to access a site they are fronting. Cloudflare offers a truly terrific service for web hosting, and we're seeing more and more of what was once the decentralized web moving behind their CDN. It's a sad thought, but that may be the shape of the future.

It's somewhat chilling to know that there are additional known problems that are still under wraps which will need to be dealt with in a fashion similar to what just happened with PHP. In the case of PHP, the news of the flaw leaked prematurely, but thankfully just two days before the planned update announcement and release. So the PHP organization was able to quickly advance their release by one day.

On one hand they were mostly ready, but exciting secrets are difficult to keep. Here's a snippet from the PHP Foundation explaining some details of the collapse of the patch news embargo. They wrote:

The coordinated disclosure of these vulnerabilities worked well until 2 days ahead of the targeted release date.

On Tuesday, October 20th, at around 15:00 UTC, the two CVE IDs began circulating on online forums and social media. They came with a wrong release time and a wrong CVSS score of 10.0 (the actual score is 9.2). No technical details leaked. In response, the PHP project brought its pre-announcement forward and published it at 19:30 UTC. It also moved the release from October 22nd, 14:00 UTC, to October 21st, 14:00 UTC. Distributors were still packaging the fix, so releasing immediately would have left many users without an easy way to update. Moving the release forward by one day was the best compromise between acting quickly, giving people time to prepare, and making sure everyone had a simple update path when the vulnerabilities were disclosed.

As a result of the leak and pre-announcement around 20:00 UTC, Calif, one of the security firms that had reported the issues, mistakenly published a video. It showed their proof of concept working against a number of targets and revealed a few additional, though limited, details about the vulnerabilities. Copies of the deleted video started circulating on social media soon after. The video contributed to accelerated development of various PoCs (exploitation proof of concepts), which then increased circulation on October 21st.

Mitigation advice we started providing to a few critical infrastructure providers around Oct 20th 23:00 UTC, may have leaked as well. The PHP security team closely monitored and inspected appearing PoCs after the leak but did not identify any functional exploits at the time of release/public disclosure.

On October 21st at 07:41 UTC, more than six hours before the coordinated release, Chainguard unilaterally broke the embargo and published built images containing the fix. Like other distributors, Chainguard had received the patch in advance so its users could update the moment the vulnerabilities were disclosed. Any attacker could now study the fix in binary form while every other PHP user was still waiting for the release. This disregarded weeks of work by many parties toward a coordinated disclosure.

It’s not difficult to imagine that Chainguard will not be offered early access to future critical release information, certainly not by the PHP Foundation and probably not by others who understand and respect the need for absolute pre-release information disclosure control.

But imagine that everything **had** gone according to plan. The outcome would not have been much different. No matter what the industry does, we’ve never had the capability of updating everything at once. So the moment PHP was announced as planned last Thursday rather than urgently a day earlier, most of the same dynamics would have ensued. Exploitation proofs of concept would have been created and bad guys would have begun attacking with the goal of compromising PHP systems before they had the chance to be patched. It’s unclear to me how days of frenzied patching and attacking can be prevented.

The well known exception to this is our web browsers. They’re able to update immediately and more or less continually. This capability was universally developed because they represent the Internet’s largest attack surface and they’ve had so many problems in the past. Once that feature was added it was self continuing. But the software update model that most of the Internet is still using today has not changed in decades because it lacked the browser’s level of

urgency. And now it's too late. In fairness, though, server infrastructure is not the same as end-user consumer browsers. So it's not fair to suggest that the model of continuous background autonomous updating could ever be applied to non-consumer enterprise or cloud environments.

One possibility that occurred to me is full stealth updating, where a horrible problem is quietly patched and resolved without the explosive fanfare and events that accompanied last week's PHP debacle. The idea would be that not until many months had elapsed and after most affected systems had been patched in the normal course of software maintenance would an "oh, by the way, we fixed this horrible problem some time ago" be mentioned. Or perhaps, even better, nothing would ever be mentioned.

Unfortunately, Cloudflare's Matthew Prince was quoted saying: *"It's going to feel like, over and over and over again, we're finding more and more bugs in very critical pieces of software. This story, which feels unique and fresh right now—in a few months, it's going to feel like we get one of these every week."* That sure doesn't sound as though whatever is going on will be suitable for any sort of quiet stealthy updating. It very much sounds as though we're in for a rough next few months.

Listener Feedback

How did this affect our listeners? Those who subscribe to receive the weekly Security Now! show notes received a heads-up about this from me Wednesday morning. I called it a "Flash Alert" and received replies such as:

Thank you, Steve! Spent the whole day in a war room today ... patching :)

This reminds me of a few stories you mentioned previously, eery to feel it this closely this time:
https://thephp.foundation/blog/2026/10/21/todays-php-security-release-and-how-were-strengthening-php-security/#leak-and-disrupted-disclosure

> The coordinated disclosure of these vulnerabilities worked well until 2 days ahead of the targeted release date.

> one of the security firms that had reported the issues mistakenly published a video

> more than six hours before the coordinated release, Chainguard unilaterally broke the embargo and published built images containing the fix

Click farmers, clout chasers, accidental reveals, broken embargos. The security community truly has to deal with a lot these days. Well, at least it got a pretty page, right? :D
https://www.php2shell.com/

Can't wait for your continued assessment of this and similar previous situations and how this might shape the future of responsible disclosure. Didn't think this is how I would write an email to you eventually. Greetings from Austria! /Best regards, Lucas Seinfeld
(Web Software Engineer for over 25 years, proud SpinRite Owner)

A short one was *"Thanks Steve! Just got my business servers patched!"* Another listener, Corey Austin who described himself as *"A Fellow Bipedal Carbon Based Life-form!"* wrote:

> *Hi Steve! I literally had been talking about this event with a customer, AS your alert email came in. Specifically, I said "Yep, this will be in the Security Now podcast next week"! What synchronicity!?! Can't wait for next week's episode! Cheers, Corey*

Another listener replied: *"Thanks for the notice. We had a hair-on-fire patching session this morning to get everything updated."* And Jason Bache of NerdsLimited.com replied: *"Thank you Steve! You made the right call throwing this one wide."* There was *"Thanks Steve! Bit of a shock to wake up to here in Australia."* Dave Mitchell wrote: *"Thanks Steve. We're already all over it (and our clients are all over us !!). High profile one, this. /Dave"* Andrew Johnson replied: *"Thank you, Steve. I know you're usually hesitant about sending out of cycle emails but this is important."* Andrew's right. I'm hesitant, because I don't want to be a source of additional unwanted noise. Consequently, my out-of-cycle emails have been very few and far between. But I received nothing but strong encouragement in this instance and it sounds as though there may be cause for a few more of these in the coming months. It also sounded as though most of our listeners were already well aware of the event, but I did receive a note that made me glad I hadn't resisted. One listener wrote:

> *Thank you Steve!! You really saved our asses on this one. I truly appreciate this emergency warning, and I'm so glad I'm on your mailing list. I'm a long-term listener; I've been with you since Day 1, some 21 years ago - I'm also a Spinrite owner :) I feel like in the age of AI (of which I'm deeply involved now) this kind of alert is going to become the new normal. Keep up the great work on the Podcast! Your friend, Scott.*

Another different sort of note was:

> *Thank you for that. I was hoping my little forum for a sports association (not search-engine indexed) was too old, but it's on PHP version 5.6. Oh well, I'll take a static archive and shut it down, it's not worth the effort to update! -Alastair*

And then there was:

> *Hi Steve, as I write this I'm in the trenches updating a client's super old (and unfortunately highly customised) PHP CRM system that's currently stuck on PHP7.4 & MySQL 5.7. Had another client message me last week before all of this asking "why are they attacking us? It seems nonstop!" I explained "it's not personal, it's not just you. They do it because they can"*

Another listener of ours, Michel wrote:

> *Hi Steve, Long-time Security Now listener from Belgium. The PHP2Shell situation this week reminded me very much of something you have said many times about legacy code and forgotten components quietly increasing the attack surface.*

I run a small personal Plesk VPS. When the PHP2Shell warning appeared, I immediately checked my only PHP website. It was already running the patched PHP 8.5.12, so at first sight everything looked fine. It wasn't. Plesk, itself, turned out to be running its own embedded PHP through sw-engine, which was still PHP 8.4.25 and vulnerable. Plesk 18.0.81 Update 4 eventually moved that to the patched PHP 8.4.27.

While checking the server, I then discovered another nice example of accumulated attack surface: Plesk had about 25 PHP handlers available across several PHP generations, although my actual website needs exactly one: PHP 8.5 FPM.

I started removing the unused versions. That exposed another hidden dependency: Roundcube webmail still requires PHP 8.3. So after cleaning things up I can now actually account for every remaining PHP runtime:

PHP 8.5.12 → my website
PHP 8.3.36 → Roundcube
PHP 8.4.27 → Plesk's own sw-engine

Everything else is gone or disabled. What struck me, was that none of the individual decisions that created this situation were particularly bad. Old PHP versions and handlers had simply accumulated as the server evolved. Until this week they were just "stuff that's still there." Until suddenly, every forgotten runtime became relevant. It felt like a very Security Now example of a simple principle: software you no longer need is not merely clutter. It's an attack surface you may eventually forget you even have.

Thought this might be an interesting little real-world anecdote for the PHP2Shell discussion. Greetings from Belgium, Michel

Yep. Things are a bit of a mess with PHP at the moment. Michel's experience is a perfect example of why it may be less easy to thoroughly clean up this PHP mess than is immediately apparent. It's not difficult to imagine bad guys who see a large potential payoff, taking the time and trouble to determine where older versions of PHP might be hiding while still being exposed to the outside world. This one may be with us for a while.

Christy Ramsey shared a perfect example of what my picture of the week caption labeled "an artifact of LLM technology." He wrote:

Hi Steve! Listening for years, we use a Spinrite Consultant license at the non-profit computer refurbishing shop where I work. It has i's own computer with SATA power and data cables hanging out of it to attach drives.

I use Claude to help write answers to questions from clients and my radio show. It recently called me out for saying 1,000 Microsoft Patch Tuesday updates a month, I pushed back and here is its response:

> I was wrong, and I need to own that one visibly. I called "1,000 updates a month" a joke and said not to let it stand as a fact. It is a fact, and it is understated. Fact, confirmed from multiple sources: Microsoft's October 13, 2026 Patch Tuesday fixed a record-breaking 1,411

flaws, including one actively exploited and two publicly disclosed zero-day vulnerabilities. According to Microsoft, the latest updates resolve 1,413 CVEs, including 1,102 affecting Windows, 134 in Office, 61 in developer tools, and 36 in SharePoint Server. And it was not a one-off: the October update follows another unusually large Patch Tuesday in September, when Microsoft addressed more than 970 vulnerabilities.

Where my reasoning went wrong: I answered from memory. Patch Tuesday counts I knew ran roughly 60 to 180 a month, so four figures sounded like exaggeration, and I did not search before dismissing it. That is exactly the failure mode my own standing rule is meant to catch: verify before correcting someone. The lesson I am carrying forward is that a number that sounds absurd is a reason to search, not a reason to doubt the person who said it.

I enjoy the show. It's an interesting time to be in computer support. Back when I started, we called Microsoft and they charged per call. So we would have rotating techs keeping them on the line to get a server up. It was well over 12 hours. I left after 5 pm and the call was still going when I came back at 9 am. I didn't hang around the team working on it, they were getting grumpy. Then forums appeared, then web sites and search, then online chat support, now AI. May you live in interesting times indeed.

Over in GRC's AI newsgroup we've been having some discussions about intelligence versus consciousness. A valuable participant there argues that we cannot claim that current LLM technology is not conscious. He asked me: *"What would it take to convince you an AI is conscious and not simulated conscious? What would be the difference between simulated consciousness and real consciousness?"* After giving that question some thought and replied:

My threshold for that determination would be a demonstration of actual understanding of the subject matter. For me, the biggest "tell" that we're all dealing with a powerful language model is that its occasional confidently-stated mistakes demonstrate that there is nothing behind the curtain that in any true sense knows what it is saying.

What has happened during the past three years, since the emergence of ChatGPT and conversational large language models, is that they have grown larger, and those creating them have worked tirelessly to more deeply hide their inherent limitations. Without question (and with the brute-force expenditure of truly astonishing quantities of cash) LLM's use of language to simulate a sentient entity has dramatically improved — no question about it. But that's all that's happened. The simulation has been polished.

I believe we will limp along using LLMs -- because that's all we have for the moment -- until we work out the right way to create a sentient machine. Large language models alone, no matter how large they become, are not it. Scaling them, scales the quality of the simulation, but that cannot change the reality of its character. What an LLM is, is an incredibly tantalizing simulation of what it MAY be possible for mankind to create. And I believe we're likely to, because so much money and livelihood has now been committed to this effort. Our ability to do so is not a foregone conclusion, but without the false promise of LLM technology we would today have the scale of compute available, nor the required desperation, that might make it possible.

A Strategic Plug Pull

Or, Discretion is the better part of valor.

Before we leave this topic, which has rightfully dominated today's podcast, I want to make one final observation. One of the prominent responses to last Wednesday's PHP surprise was to temporarily pull the plug on PHP-based Internet services. This appeared to be mostly a European response, likely due to a neighbor effect, but it immediately put me in mind of Russia.

It's never been entirely clear why Russia has seen fit to go to the trouble of developing the means to quickly sever all external global networking ties and switch to a totally internal network. But for several years past we've been covering their several tests of that capability. It's a bit creepy since one motivation might be to prevent cyber-reprisals in the event that they do something that upsets the rest of the world. I'm hoping that it's just a defensive measure. But either way, the notion of pulling up stakes, disconnecting from anything and everything cyber and hunkering down is an interesting strategy. There is some history of this having been done in the past.

We all recall the Heartbleed incident, where it was discovered that chunks of raw internal web server memory could be exfiltrated over a secure connection and proofs of concept managed to extract a running server's private TLS web certificate. It turned out that wasn't all that could be extracted. In response to the urgency of Heartbleed, Canada's tax agency took its tax-filing website offline as a preventive step just three weeks before the national filing deadline — meaning that the timing was quite inconvenient. Those suspended tax services remained offline for four days, from April 9 until April 13, and Canada's filing deadline was extended to May 5 to compensate. That was 12 years ago, back in 2014, when pretty much everyone's personal data was not already in the hands of the bad guys. So the fact that 6 hours before Canada's service suspension 900 Canadian Social Insurance Numbers were "bled out" through Canada's tax service server was a bigger deal then than it would be now. Nevertheless, other Canadian federal departments were vulnerable too and they wound up being shut down two days later.

In the US in 2020, on a much smaller scale, the SolarWinds Orion mess caused CISA to order federal agencies to power down their vulnerable servers. And this happened again the next year in 2021 with the Microsoft Exchange Server ProxyLogon vulnerabilities. Better to just unplug when a known problem may be exploited before it can be patched.

Then we had the Log4Shell panic that swept the Internet world shortly before the end of the same year, 2021. In a reactive panic, Quebec shut down 3,992 government websites, including the education and higher-education ministry sites. The minister made the point that without an inventory, confirming which sites even used the library could take days. He compared it to checking every room in every government building to find out which ones use 60-watt bulbs. So, better safe than sorry. More broadly across Canada, the Canada Revenue Agency, Employment and Social Development, and Toronto's Metrolinx transit system also took sites offline.

Also on a smaller more local scale, the reaction to the Ivanti Connect Secure flaw was CISA ordering every US federal civilian agency to disconnect all Ivanti Connect Secure and Policy Secure VPN appliances within two days of the order. That directive applied to 102 federal agencies which researchers believed should have simply patched rather than disconnecting.

So, there's precedent for implementing **a strategic plug pull** — if only for an hour or two — when things look bad enough to warrant it. Last week, the cybersecurity agencies in Europe, Australia, Japan, and Britain all told their PHP users to either update immediately or take their sites offline until they could patch. Pull the plug until it's fixed. This contrasted with earlier pull-the-plug responses which were usually decisions made by individual governments or agencies. In the Netherlands, many organizations followed the National Cyber Security Centre's example and pulled their own plugs, going dark, including the Port of Rotterdam, the government portal Overheid.nl, and all five of the country's largest political parties, and several hospitals.

I am putting this **strategic plug pull** idea front and center because it's a strategy that might be worth keeping on hand and in mind. Although it can feel like overreacting, and perhaps it would be, depending upon the specific circumstances in the moment, it would likely be better than suffering an intrusion. We have no idea what Matthew Prince and others are aware of that we are not. We only know that, for a change, they are ***not*** saying *"something is likely to happen"* — this time they're saying *"we have already discovered and know of several things that are definitely going to happen, but we're not yet ready to drop those bombs on the world."*

Last Wednesday morning when I learned of the PHP trouble, I immediately knew that GRC's forums were vulnerable. I also knew that whatever happened my longstanding inherent mistrust of PHP would protect the rest of GRC since, as I noted earlier, that server's blast radius has always been deliberately contained. So I did not panic. I first applied the quick PHP.INI configuration mitigations that had been provided, then I installed the latest branch version and migrated all of GRC's PHP usage over to it. The same branch version migration meant that only the PHP binaries needed to be updated. No other configuration changes were needed. And realistically, compared to just about anyone else on the Internet, I'm a low-value target so I did not imagine that anyone would be coming after GRC immediately. Not panicking has its place.

But that said, none of us have any idea what those next and now-anticipated surprises might be. I'll be quick to take my own advice to strategically pull the plug on GRC's Internet connection if any of what happens next turns out to warrant it. After playing dead on the battlefield to avoid being killed, Shakespeare's Falstaff famously said *"Discretion is the better part of valor"* is the one to keep in mind. No one gets any points for remaining online in a cyberstorm if that results in an intrusive compromise.

In these times of unprecedented cyber unrest, I promote the formalization of "The Strategic Plug Pull." While no one likes an administrative service outage, the bad guys will almost certainly be unable to attack anyone whose attack surface is offline. And consider that any event that would be large enough to cause you to pull the plug to take your or your enterprise's services offline would be big enough news that you would not be alone, or would at least be understood.

I'm not a doomsayer but we're surrounded by those in the know who are saying that there **will** be some rough times ahead. Perhaps we won't be affected. We won't know until we know. But as I said: No one can attack an attack surface that's offline. So, finally, if a bit of preparation for the possibility of strategically pulling the plug would make the decision easier and less painful, now might be the time to prepare for that. Everyone's situation, responsibilities and reporting structures are different... but perhaps give this idea some thought.

The eMudhra Mess

Finally, we need to talk about **the eMudhra mess** which broke last Tuesday, the day before the PHP debacle. It was an utterly insane event that I would have normally gleefully pounced on as this podcast's topic of the week — if even it had not been overshadowed by PHP, which it was.

The private keys used by the universally trusted intermediate certificates used to sign the TLS and code signing certificates issued by eMudhra, a Certificate Authority based in India, had not only leaked — which would have been bad enough — but those private signing keys had truly inexplicable values. This created a scandal so big that there was no recovery from it. Consequently, eMudhra is no longer in the certificate business.

I'm going to let ArsTechnica's Dan Goodin fill in the details. He wrote:

> *Security personnel around the world are on high alert Tuesday following the discovery of eight TLS certificates with private keys so weak they could allow anyone to mint trusted browser and app credentials for any site on the Internet.*

And I'll add, also sign any code. But Dan's 100% correct about the impact of this. He continues:

> *As of late morning Tuesday, browser makers and some major apps and programming languages had blocked all eight of the intermediate certificates. There are no reports of certificate transparency logs, which track the issuance of all browser-trusted certificates, showing the compromised keys being used maliciously. The threat remains far from over, however, because it will be hard and time-consuming to update other key parts of the Internet, such as Linux distributions, embedded and Internet of Things devices.*
>
> *One word: bonkers*
>
> *The eight intermediate certificates belong to India-based certificate authority eMudhra Limited. An informal advisory published to emudhra.we-have-a-problem.com on Tuesday posted the message "We have a problem here – Wayne." Those words were digitally signed using the private key of one of the eMudhra certificates. Armed with proof that the reporting party had control of the key, defenders scrambled because the same key could be used to sign unauthorized certificates that would be trusted by millions of server clients worldwide.*
>
> *Filippo Valsorda, a cryptography engineer and a cryptography maintainer for the open source Go programming language, told Ars: "This will be a massive cleanup job, bigger than Heartbleed. There will be a very, very long tail, IoT devices, Docker containers, forgotten Linux machines, unmaintained CLI tools that shipped their own copy of the roots (which is common in some ecosystems, like Python). Also, Linux distributions are still rolling out updates, some of them are not ready yet, and then system administrators will need to apply them to a lot of machines."*

I'll pause again to put a bit of a point on that. As we are going to see, the truly inexplicable nature of the discovered private keys that eMudhra's own root certificate signed, and that Sectigo also cross-signed, means that any device that carries its own internal root certificate store without some provision for that store being updated — which means anything with firmware that needs to generally connect to Internet resources — will now and forever fully trust

any certificate it might receive from anything and anyone that's signed by one of those now utterly compromised intermediate certificates. It really is a mess. Dan's reporting continues:

> *Heartbleed is the name of a critical vulnerability in the OpenSSL cryptography library. In 2014, it came under mass exploitation. Exploits let many unaffiliated hackers decrypt TLS-protected exchanges between parties. They also recovered private TLS certificate keys, session cookies, and passwords. The attacks compromised thousands of servers, some belonging to large companies and government agencies around the world.*
>
> *Recovery required a long series of steps that took years. Steps included patching OpenSSL on all affected devices, reissuing millions of TLS keys, revoking old certificates, rotating passwords and session tokens and updating embedded devices and other forgotten systems —something that was extremely time- and resource-intensive, especially at Internet scale.*
>
> *Valsorda explained: "Every time you need a lot of different people in different contexts to do something, it will take a long time. This time it's worse because there are more contexts, more people, and more heterogeneity."*

And here comes my favorite part which Dan phrases poetically, writing:

> *The compromised eMudhra keys were defective at a level that's challenging the imagination of many of the engineers studying the incident. A normal NIST P-384 ECC private key contains roughly 384 bits of entropy, a size allowing almost 2^384 — a number with 116 digits — different possible keys. The private key sizes of the compromised keys ranged from 3 to 8 bits. Astonishingly, the private keys were 4, 56, 66, 115, 155, 156, 230, 237.*

Because this is utterly unimaginable in this day and age, I want to make sure everyone understands what Dan just correctly described. We're all used to seeing a typical cryptographic key, right? It's some ridiculously long jumble of decimal or hex digits that would be impossible to remember, let alone to even correctly write down without making a mistake. In the case of a NIST-standard P-384 bit key, it would be 116 decimal digits.

Against that backdrop, one of the private keys that the entire world was trusting until last Tuesday, and much of the non-updatable embedded world will continue trusting forever, was the single digit number "4". Not some jumble of 116 digits. Just "4". And that others, as Dan noted, were 56, 66, 115, 155, 156, 230 and 237. It's true that a private key can be anything, and that a private key of "4" is every bit as valid as one having 116 digits, rather than 1. But something must have gone very wrong somewhere for this to have happened. Dan's story explains:

> *Valdorda said: "That is extremely in the 'this should not be possible' category. Maybe some horribly dangerous key generation interface has a 'size' parameter and they set it to 1 by mistake???" Ben Cartwright-Cox, director of BGP.tools, said: "My guess is that it's example code that accidentally leaked to production, but hard to know. But it's bonkers that no one noticed."*
>
> *At this early stage, the most immediate threat appears to be for those using VPNs, especially in countries with governments likely to maliciously intercept traffic. There are likely many*

other cases for concern. End users largely have no recourse other than to hold off using many non-browser forms of TLS communication until they receive assurances updates are in place.

As noted, most browsers have already blocked the intermediate certificates. The exception appears to be Edge running on Windows 10, which at the time this article went live was still connecting to 12 vulnerable hosts. In response to questions asking why and if the company has plans to do so, a Microsoft representative said the company will be in touch if it has anything to "share." Meanwhile, Signal, WhatsApp, Wireguard, and other apps that don't rely on TLS for confidentiality were never affected. The emudhra.we-have-a-problem.com site show apps known to be patched so far.

The incident is the latest to expose a fatal flaw in the TLS regimen. It's only as strong as its weakest link, which in this case was eMudhra. Fortunately, transparency logs account for much of that failure, but this measure requires parties to actively monitor massive feeds and works only after unauthorized certificates have been issued. eMudhra representatives weren't immediately available to answer questions.

Dan then finishes his piece by somewhat understating the consequences. He writes:

One caveat: exploiting compromised certificates requires an active adversary-in-the-middle attack, meaning the attacker must act as a proxy between two TLS parties and receive encrypted traffic, decrypt it, reencrypt it, and send it to the other party. Such capabilities, in some cases, are usually limited to only the most well-resourced attackers. People with control of a local network—at a coffee shop, university, or other organization—face much less of a challenge.

What Dan described is true. But on-the-fly traffic interception is only one way of abusing a compromised certificate. The gift that the bad guys have been given with this eMudhra debacle is the ability to create their own fully trusted certificate for any TLS or CodeSigning use. This means that they don't need to perform a complex traffic intercepting man-in-the-middle attack. They can much more easily impersonate any web site and server on the Internet. They can create a fake site for any domain they wish. The good news today is that no one's web browsers will fall for that anymore. But the bad news is that there are many other instances where, as those Dan quoted for his piece, said that the static trust carried inside embedded and IoT devices may be subject to abuse lasting their remaining lifetime.

I've included by far the most useful link I've found for answering any additional questions about the current state of this mess. The site's creator/owner, Brian Vitko notes at the bottom of the page in question: "Built by someone who actually had to push Chrome to 5,000 endpoints on a Tuesday." — meaning last Tuesday: https://getrff.com/blog/emudhra-compromise-linux-servers/

Brian's "GetRFF" site is the home of some neat looking Windows remote management tools that our listeners might also find interesting. I like this guy's style a lot. https://GetRFF.com