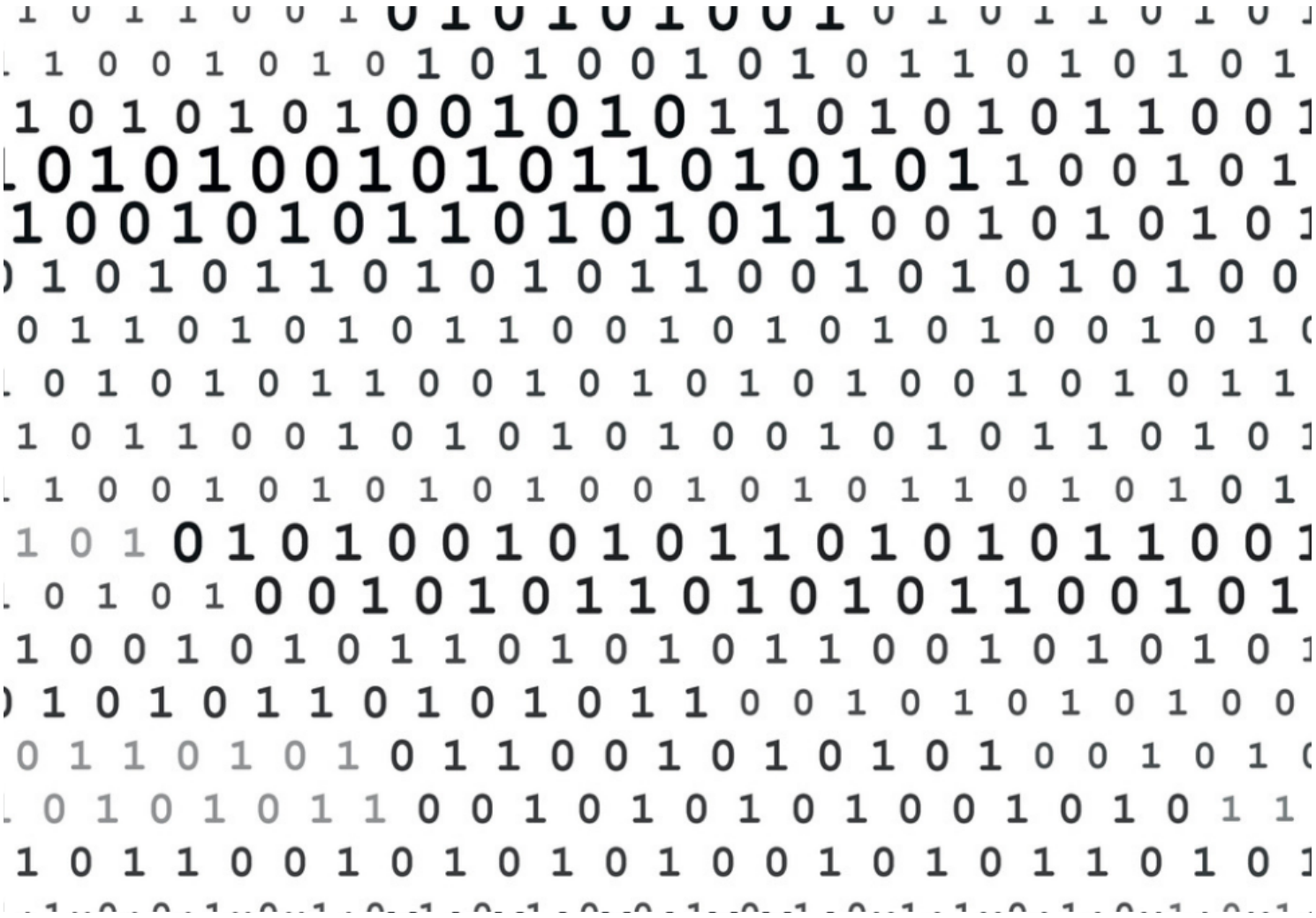

Why GitHub Won't Protect PHP

Posted by: Charlton Trezevant 6 min read

Times have changed, and hosting is just one part of an evolved threat model.

Recently, we saw the widely-publicized breach of the PHP project's development infrastructure†. Two malicious commits (1,2) were introduced into the PHP interpreter master branch using the compromised identities of core maintainers, before being caught and reverted by the PHP team the next day.

As with any attack against a large open source project, this news was the source of much consternation among developers and security professionals alike. The PHP maintainers' response was swift, and the implementation of several new security measures were announced on the PHP internals mailing list in response to the breach:

- The PHP project's Git server at git.php.net is being fully decommissioned. Instead, the entire project will migrate to GitHub.

- Any contributors with write access to PHP repositories must be part of the PHP GitHub organization and enable two-factor authentication on their GitHub account. The PHP project had previously built its own reputation-tracking access management system for contributors, which is being deprecated.

In summary: Move the project to GitHub, lock write access to organization members… Done. But is that really enough?

Migrating to GitHub certainly isn't a bad move and has benefits aside from security, such as streamlining community engagement. However, the notion that migrating to a large hosting service will fix the fundamental problems leading to PHP's compromise is disingenuous, even dangerous, and leaves critical improvements to the project's security posture unexamined.

Today, the software development and security landscape has evolved greatly:

- The tools available for protecting the integrity of development processes have grown in both capabilities and complexity.

- The threat model of large open source projects now includes malicious actors with sophisticated tools and techniques (not to mention state backing in some cases).

- The economies of the software ecosystem as a whole, where a handful of core packages are used in more places than ever, means that researchers get paid a lot of money to find and sell bugs, rather than contribute fixes.

Let's explore that in more detail.

We've Been Here Before

Supply-chain attacks are all the rage following the SolarWinds breach in late 2020, but these types of threats are certainly nothing new. In fact, a strikingly similar attack took place against an even higher-profile open source project roughly one decade ago, with the 2011 breach of the Linux kernel source repositories.

While the breach didn't lead to the insertion of any backdoors in the kernel source, it did cause the Kernel team to re-think their threat model:

> Ever since the 2011 compromise of core kernel.org systems, **the main operating principle of the Kernel Archives project has been to**

> **assume that any part of the infrastructure can be compromised at any time**. For this reason, the administrators have taken deliberate steps to emphasize that **trust must always be placed with developers and never with the code hosting infrastructure, regardless of how good the security practices for the latter may be**.

Excerpt from the Kernel Maintainer PGP Guide.

You may have noticed that a major infrastructural overhaul was **not** part of the Kernel team's response to the attack. In fact, the Kernel project continues to own and operate their own Git repositories just as they did in 2011. Why?

For one thing, Git provides a high degree of cryptographically-backed tamper resistance. This means that any attempts to directly modify repository contents directly (i.e., objects or packfiles) will be detected by Git, regardless of who's hosting the server:

> Every single piece of data, **when git tracks your content, we compress it, we delta it against everything else, but we also do a SHA-1 hash, and we actually check it when we use it. ‡**
>
> […]
>
> **[Y]ou can trust your tree, all the way down, the whole history.** You can have 10 years of history, you can have 100,000 files, you can have millions of revisions, and you can trust every single piece of it.
>
> […]
>
> I guarantee you, **if you put your data in git, […] you can verify the data you get back out is the exact same data you put in.**

Excerpt from Linus Torvalds' 2007 talk on Git's design.

The ability to detect corruption is a vital feature of any good source control system, but it leaves one key security problem unsolved: Identity. Anyone with write access to a Git repository can attribute a commit to another user without providing any further authorization (yes, even on GitHub). Without commit signatures, there are no strong guarantees of identity in Git that would prevent an evildoer from attributing malicious changes to a trusted author.

In the case of both the Kernel.org and PHP breaches, the underlying issue wasn't that the project's servers were accessed by a malicious actor. Rather, the attackers were able to directly commit new changes by co-opting the identities of trusted maintainers. It's for this reason the Kernel team embraced a distributed trust model centered around commit signing by individual developers, rather than basing their root of trust on a single large hosting provider.

Thankfully, GitHub features built-in support for commit signing and validation, and the PHP project can take advantage of extended GitHub features to ensure any new commits pushed to their repositories are signed and verified using contributors' trusted keys.

Commit signing isn't the only solution, of course. Integrating static analysis tools into PHP's code review process and CI/CD pipelines could add another line of defense against malicious changes. Along with catching common classes of bugs early on, these tools can be used to automatically flag changes containing "dangerous" functionality (such as new paths to zend_eval_string) for further review.

The Bigger Picture (Spoiler: It All Boils Down To Money)

There's another unique element to the PHP breach that warrants further discussion: the malicious commits (1,2) mention Zerodium by name.

While there's no indication that Zerodium was behind the attack (the CEO of Zerodium has tweeted that the breach was the work of a "troll" unaffiliated with the company), there's a larger conversation that needs to take place about the 0day market: What's to be done when the line is crossed from private vulnerability research into something much darker?

Let's consider the worst-case scenario and operate with the assumption that the PHP attackers were motivated by a desire to sell these backdoors on the open market. Had these shipped in a PHP release, would Zerodium have purchased them? A policy regarding the eligibility of vulnerabilities derived from these kinds of techniques wasn't clearly defined on Zerodium's Program Overview or FAQ.

As a community, we have a responsibility to confront the incentive structures created by the 0day industry. Even though Zerodium hadn't carried out the attack itself, it's possible that the business model of companies like Zerodium have created a financial incentive for these types of attacks to occur.

Purchasing 0days found through legitimate research is one thing, but rewarding malicious actors for these types of breaches is tantamount to criminal solicitation. Without a clear policy on ethical purchasing guidelines, attackers may be incentivized to copy the methodology of the PHP breach and actually sell the vulnerabilities. Or a core maintainer of some important project might make their retirement off of a few intentionally careless lines. With payouts of up to $2.5M on the line, there's a significant financial incentive behind these sorts of scenarios.

To be absolutely clear, none of this is to imply that Zerodium is engaged in any criminal activity whatsoever. But in the interest of avoiding a kind of total war, there's a genuine need for a larger discussion about ethics and regulation in the 0day market.

TL;DR

Moving to large hosting platforms isn't a complete solution for hosting code and development workflows in a reasonably secure way. Without additional steps, migrating to GitHub doesn't solve other key weaknesses that ultimately allowed malicious commits into PHP core.

There's more to this conversation. Here are the main takeaways:

PHP developers should:

- Recognize the responsibilities of the PHP project as maintainers of a platform powering nearly 80% of websites in 2021. In short, **that means a serious and in-depth commitment to security** at more levels than infrastructure alone.
- Update the git workflow for core PHP maintainers to include signing and verification of commits using Git's inbuilt GPG facilities. Ideally, dedicated hardware devices would be used to protect developers' key material.
- Implement static analysis tooling into PHP's code review process and CI/CD pipelines. In addition to catching bugs, these can automatically flag changes containing "dangerous" functionality (such as new paths to zend_eval_string) for further review.

Zerodium should:

- **Denounce the methodology** used to introduce this weakness into PHP core outright.
- **Commit to a policy** of not purchasing bugs introduced through direct attacks on open-source projects to avoid incentivizing criminal behavior.

Closing Thoughts

As security professionals, we're part of a larger community representing a diverse range of interrelated, interdependent organizations. Fundamentally improving the state of security in our industry requires attention to the human element- particularly, drawing upon our own expertise to improve developers' awareness of security problems and access to educational resources, tooling, and mitigation techniques. Organizations like the Linux Foundation's Core Infrastructure Initiative and Google's OSS-Fuzz represent strong examples of new and holistic approaches to improving security in the open source community.

Those in the security industry who are concerned by the PHP breach should consider actively contributing to open source projects in some way, whether that's organizing awareness campaigns, contributing documentation, or simply by voicing an informed opinion. Through our collective efforts to share our expertise, perhaps the skills and practice of our craft will become available to many more developers.

---

† In an update shared by the PHP maintainers, the root cause of the breach was determined to be a weakness in the PHP project's central authentication system. According to the post, the attackers were able to dump a credential database which stored passwords using a weak password hashing scheme (MD5). Most likely, cracking these ultimately granted write access to the PHP repositories.

‡ Astute readers might be aware of published research on SHA-1 collision attacks. In response to these vulnerabilities, Git is in the process of transitioning to SHA-256. GitHub has also rolled out protections and published tools to mitigate collision attacks.

Categories: APPLICATION SECURITY BLOG CYBERSECURITY

CHARLTON TREZEVANT

APPLICATION SECURITY CONSULTANT,
GUIDEPOINT SECURITY

Charlton Trezevant, Application Security Consultant at GuidePoint Security, is a software developer and security consultant residing in St. Petersburg, Florida. Since 2015, Charlton has drawn from years of experience in multiple technical domains, using his skills to great effect as a security professional, mentor, researcher, and award-winning participant in national cybersecurity competitions.

Charlton has led and participated in vulnerability assessments, consultation, and penetration testing for industries including higher education, entertainment, finance, and technology. His experience includes application and network penetration testing, security auditing, threat modeling, social engineering, network architecture, software development, and curriculum design.