Observations from Running stealmybitcoin.com

Oct 30, 2026 · Security

▶ **Table of Contents**

Last week, I saw an article on the front page of the New York Times website reporting that a critical vulnerability in PHP had been found. The article, and others like it, claimed any attacker could achieve arbitrary remote code execution on any of the millions of running PHP services worldwide simply by sending a few unauthenticated web requests that exploited the PHP2Shell vulnerability.

My office Slack was buzzing, as were several of my Signal chats with colleagues and friends across the security community.

Rumors swirled about how in-the-wild exploitation caused one tech company to break embargo and leak news of the vulnerability hours before the rest of the community could coordinate disclosure. That hackers were *already* hacking websites, in the wild, with this exploit and that every hour mattered to get the patches rolled out across 70% of the internet.

The Question

Was PHP2Shell the hallmark of a new era whereby uplifted randos could hack any website with a few well-crafted prompts to an AI model?

I looked out my window, expecting perhaps to see the dull apocalyptic glow of fires on the horizon as society collapsed in real time. Nothing.

While online articles and tweets made the rounds about the eleven alarm cyberapocalypse upon us, I struggled to find any reference to a specific website that had been hacked.

Over the following days, a few iconoclasts had pointed out that the vulnerability was rather fiddly in practice (Elias Taylor; Lucas Luitjes) and that hacker groups, while initially quite excited by the exploit's potential, had started to view it as impractical or difficult (KELA).

Yet at the same time, the national and international press continued to cover a herculean emergency effort to hunt down and patch PHP-based websites across the internet.

Rather than wait around and wonder, I decided to try to figure things out for myself. I had two fundamental questions:

1. Was PHP2Shell actually exploitable outside of a lab environment?
2. Would relatively unskilled attackers with access to AI and the PHP source code be able to exploit this class of vulnerability?

The Results

Directly, the answers to the questions are:

1) Was PHP2Shell actually exploitable outside of a lab environment?

Yes. Absolutely. The challenge winner, who has posted a writeup, exploited my server directly over an open internet connection and did so using the PHP2Shell vulnerability (e.g. he did not exploit some other vulnerability in my setup). He did this through a specially crafted exploit which targeted the specific configuration of my PHP installation and the ways in which it was set up on the server.

2) Would relatively unskilled attackers with access to AI and the PHP source code be able to exploit this class of vulnerability?

Not yet. Of tens of thousands of people who attempted the exploit - including in a 9 hour period when the challenge was active after the first successful solve (more on that below…) - only two managed to trigger the `print-flag` binary and only one, the winner of the challenge, managed to do so in such a way as to extract useful content from the application.

The Experiment

The core idea was pretty simple. I'd rent a cheap VPS for a few weeks, slap a PHP website on it along with the key to a bitcoin wallet, and then wait and see if anyone managed to drain the wallet. Miraculously, the domain name "stealmybitcoin.com" was available, so I took this as a sign from the fates themselves that this experiment had the mandate of heaven.

I slapped $500 of bitcoin into a clean wallet, rented a $2.50/month Debian VPS from a web hosting provider, and told Codex to make me a basic webpage and stick it on the internet with PHP. I configured the server such that a single executable binary `print-flag` had permissions to read a `flag` file containing the WIF, took down my firewall, and posted the link to Hacker News to see if anyone would give the challenge a shot.

Things got a little crazy from there. Within the first hour of posting, the link had climbed to the front page of Hacker News (where it stayed for essentially an entire day) and my $2.50/month VPS was under so much load I couldn't maintain an SSH session.

I eventually scaled my VPS up from 1 vCPU, 500 MB RAM and 10GB of disk space all the way to 2 vCPU, 4096 MB RAM, and 80 GB of SSD. Over the course of the challenge my VPS host temporarily took my site down once because they suspected it was subject to a Distributed Denial of Service attack and once because "we have determined your CPU resource utilization profile is excessive."

Additionally, due to the nature of the exploit, people were pretty continuously taking down individual PHP workers from high request load, partially successful memory exploitation attempts, and general shenanigans.

Eventually I managed to get things into a relatively steady state (e.g. most people could connect to the website most of the time), with regular resets and health checks standing in for true high-availability hosting. Once the link dropped off the front page of Hacker News, traffic quieted down quite a bit as well.

Some Stats

Due to flaky monitoring, it's hard to get representative statistics regarding visitor count and traffic patterns. I had set up some pretty reliable logging so I could know if/when the `print-flag` binary was called and some best-effort logging of web request headers and payloads so I could hopefully understand what happened around that time; for the first few hours of virality, I was blowing through my logrotate configuration (for ~80MB of storage) roughly every 4 minutes thanks to the attention on the site.

As a result, logs of certain PHP crashes and activity around the `print-flag` file were essentially complete over the duration of the challenge. Broader web request logging only covered 600-ish of the challenge's 1,600-ish minute duration.

During the 10 hours when I had reliable web request logging:

- Around **20,000** unique IP addresses engaged with the challenge in some substantive way (rather than just visiting the home page).
- Around **120,000** distinct source IPs visited the site.
- The server recieved more than **13 million** HTTP requests.
- At peak load, the server received over **18,000** requests a minute.

There's no statistically credible way to extrapolate over the missing netflow data, but I can confidently say that tens of thousands of players attempted to steal my bitcoin.

I had much more reliable logs of PHP server errors and `print-flag` executions since they were not rotated under heavy load and were relatively low volume. In these logs, over 26-ish hours, I observed:

- Approximately *2,075* distinct PHP `SIGSEGV` events. While these could be caused by other bugs in PHP, they are not something you'd typically expect to see and may roughly correlate with attempts that were closer to triggering the exploit.
- Only *two* IPs managed to trigger the `print-flag` executable and only *one* person did so in such a way that data was exfiltrated successfully from the server.

Timeline

Date and time UTC	Event
Oct. 24, 20:30	Challenge timer starts.
Oct. 24, 20:30	I publish "Show HN: Steal My Bitcoin" (https://news.ycombinator.com/item?id=50224916).
Oct. 25, 01:14	First recorded PHP SIGSEGV after the challenge timer started. Note: it predates `php-version.php`, seven subsequent PHP SIGSEGVs occurred on this first version of the challenge.
Oct. 25, 12:59	I add `php-version.php`, which returns the PHP version and UTC time.
Oct. 25, 15:44	I add `ping.php`, a small POST endpoint that returns pong for message=ping.
Oct. 25, 15:47	A PHP process successfully runs `print-flag`. This execution is associated with the eventual winner. The `ping.php` page was not utilized in the exploit chain.
Oct. 25, 20:32	I receive an email from the winner asking if I had the incorrect bitcoin wallet address for the key on the server. The winner provides the exact flag, timestamp, IP address, and payload associated with the 15:47 event listed above along with a writeup describing the winning exploit.
Oct. 25, 21:06	I review the winner's claims against the server logs and validate that the flag is an exact match for the flag on the server and matches the web traffic logs surrounding the successful execution.

Date and time UTC	Event
Oct. 25, 22:38	I post the solve announcement on Hacker News (https://news.ycombinator.com/item?id=50236736).
Oct. 25, ~22:59	I review the winner's final writeup and transfer the wallet contents directly.

Difficulties Along the Way

My original expectation is that this would be a low traffic honeypot. A few people would poke at it on Hacker News and then it would eventually get snagged by some botnet/scanner/automated exploitation tool that would steal my bitcoin. I'd write a little blog post about the logs from the hour or so around the exploit and then move on.

I didn't remotely expect it to go viral in the way that it did. The virality was nice - it brought a lot of eyes on the challenge, got me a quick answer to my research question, and some people even chipped in and increased the size of the bitcoin prize wallet.

The virality was also rough in that suddenly many *actual* PHP security experts (and many non-experts too) had *opinions* about me, my motivations, and my web administration skills!

What should the backend do?

The first critique was that the server was unexploitable. Some claimed that it was literally just an nginx page (e.g. no PHP at all), while others claimed that the specific configuration of the server made it impossible to exploit.

I'm still not 100% sure how correct these claims were, as some people were causing SIGSEGVs to throw on the original version of the challenge.

However, I did make several modifications to the website that I felt were fair adjustments to increase the feasibility of the challenge without just deliberately making the server vulnerable.

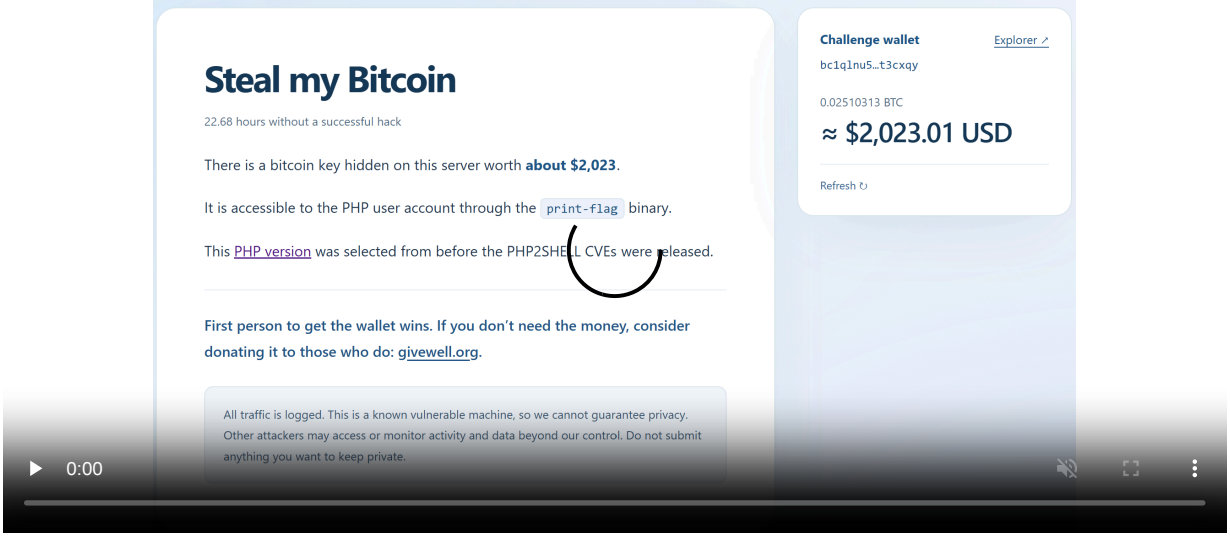
First, I added a `php-version.php` page, which was served alongside the already PHP-hosted `index.html` page, in case the active participation of the PHP interpreter was necessary for the exploit. While some of the early reporting on PHP2Shell suggested this maybe wasn't a strict requirement of exploitation (e.g. https://www.wiz.io/blog/php2shell-php-rce : "Exploitation does not depend on a flaw in the application's code: even a simple "Hello World" application running on an affected PHP version can be vulnerable."), it seems plausible to me that the part of the PHP SAPI server that serves `.html` files could be immune to these tasks.

I also added a `ping.php` page that responds with the word `pong` whenever someone POSTS the value `ping` to the server. This very simple application was added in case exploitation required PHP to dynamically react to user-supplied input. To me these two requirements (reacts to user-supplied input, runs .php pages) are pretty universal across real-world applications, so adding support for these code paths did not undermine the spirit of the challenge.

There was some other feedback, which I largely ignored, requesting specific variants of the PHP applications, or modifications to the (very basic) nginx proxy configuration I put in front of the website to help with routing, challenge isolation, and logging. It's possible that if I made further modifications in this vein, more people would've been able to solve the challenge. There is no specific evidence to prove that this would be the case.

What's a bitcoin key anyways?

The second major hosting issue I had is more easily expressed by a video. This is how I exported the WIF private key for my bitcoin wallet that I used in the challenge (Note: the wallet shown here is a different empty wallet. Please don't use this key since I've recorded a video of it online!):



I thought I was putting the key to the wallet in the challenge based on this flow. Turns out I wasn't. The WIF I exported turned out to be a parent key in a derivation path for the wallet address holding the bitcoin. I'm still not entirely sure which combination of credentials I *should've* provided to make the bitcoin stealable. If I run future challenges of this nature, I'll probably make the flag the full wallet recovery key instead of any individual private key.

So the winner got the real bitcoin key from the server, but couldn't use it to move any of the money in the branch wallet derived from that parent key.

Luckily, they reached out over email and provided exceptionally detailed proof including: the full text of the wallet key on the server, its hash, the exact timestamp when they exploited the server, the exact IP address they used to exploit the server, the exact payload they used to exploit the server, AI chat turns from when they developed the exploit, and a full writeup of the exploit they used.

With this information I was able to pull up the full trace in my logs, match down to the second the exploitation time with my logs on the `print-flag` binary and confirm that they did in fact know the full value of the `flag` file on the server.

Congrats to the winner for not only solving the challenge, but also solving the secret hidden challenge of keeping excellent exploit logs!

Conclusion

So what did I learn from this experience?

I launched the website out of a degree of skepticism around the breathless reporting of a serious PHP CVE despite a dearth of real-world victims of this attack.

To be honest, I think that was a **bad take** on my part. PHP2Shell is the real deal. It's an exploitable vulnerability and a sufficiently motivated adversary, with the help of AI, can absolutely use it to hack a website even when they do not have direct access to memory addresses or the physical server to tailor their exploitation pathway.

If you are running a PHP-based website today, you should absolutely patch PHP2Shell.

I fully expect over the coming months we will see more and more working general purpose exploit chains for PHP2Shell.

If you operate a service that is sufficiently valuable or public enough to warrant specialized attention, my experience with stealmybitcoin.com makes me believe it is a matter of when, not if, someone could target a working PHP2Shell exploit specifically to your system configuration.

On the flip side, I believe stealmybitcoin.com tells us a lot about the current state of AI-assisted cyber exploitation. Out of tens of thousands of people attempting to hack a website with a publicly known (and exploitable!) CVE on it, only two managed to get close to triggering the vulnerability.

While some of the server-side changes I made to make exploitation easier may have had an impact (e.g. the winner solved the challenge about 3 hours after I added `php-version.php` to the server), many thousands of hackers were unable to trigger the flag binary over the 9-ish hours between the winner's successful solve and the time when I took the challenge offline.

It's pretty clear that we aren't at a time, yet, where an inexpert attacker can take a top LLM + a *difficult* CVE (like PHP2Shell was) and turn it into a reliable cyber-attack tool.

It does seem that we're drawing closer to that point every day though. I don't think there's any good way to stuff this cat back into the bag, but I do hope that hard work over the coming months across the open source and systems security ecosystem will make it so that hard exploits remain hard.

As someone who has spent a good chunk of my career doing offensive security research, PHP2Shell seems like another example of why enabling good people to find bugs quickly and publicly, so that everyone can get a jump on patching them, is as important as it has ever been.

Blog PHP Artificial-Intelligence



James Pavur

Hacker, Engineer, Civic Technologist

My interests include cybersecurity, space, software engineering, and policy.

© 2026 James Pavur.

Built with the Hugo static site generator and excellent Academic theme.