mySites.guru

Hacked Yesterday, Exploited Today: Why One Cleanup Is Never the End

The first hack plants a dormant dropper. The real damage comes in the second wave, days or weeks later. Here is why monitoring beats one-shot cleanup.

Phil E. Taylor | 24 June 2026

JCE Profiles Hack · CVE-2026-48907 (24th June): Three weeks on, the JCE hack is still compromising Joomla sites, and breached sites are now being wiped. Check your site today with mySites.guru.

Read the alert >

Three unauthenticated remote-code-execution zero-days hit Joomla in June 2026, one after another: JCE, SP Page Builder and iCagenda. The patches are out, the scanners have moved on, and a few weeks later it all feels like it has settled down. That calm is the dangerous part. The first wave of attackers has done its job: it got in, dropped its files, and went quiet. Those files are still there, waiting.

This is the part most "my site got hacked" advice gets wrong. A modern compromise is not a single event you clean up and forget. The first wave is reconnaissance and groundwork. It plants dormant uploaders, hidden admin accounts and malicious scheduled tasks that do nothing visible at first. Then the second wave arrives, days or weeks later, and uses that groundwork to plant more backdoors, inject JavaScript into your template files, redirect your homepage to spam, and turn the site to profit. Being hacked yesterday does not mean you are safe today. It usually means the opposite.

That gap between the breach and the damage is exactly where reactive cleanup fails and monitoring earns its place. You cannot clean what you cannot see, and the second wave does not announce itself on the page a browser scanner checks. It changes files on disk, between scans, when nobody is looking. Catching it means watching for those changes continuously, not running one scan after the horse has bolted.

How mySites.guru catches the second wave between two clean-looking scans

A dormant dropper planted in the first wave will not change the homepage your visitors see, so a browser-level scanner that loads your front page finds nothing. What it changes is files on disk. mySites.guru's audit is built around that fact, and the most on-thesis tool in it is called **Files Modified Between Audits**: it compares every file in this

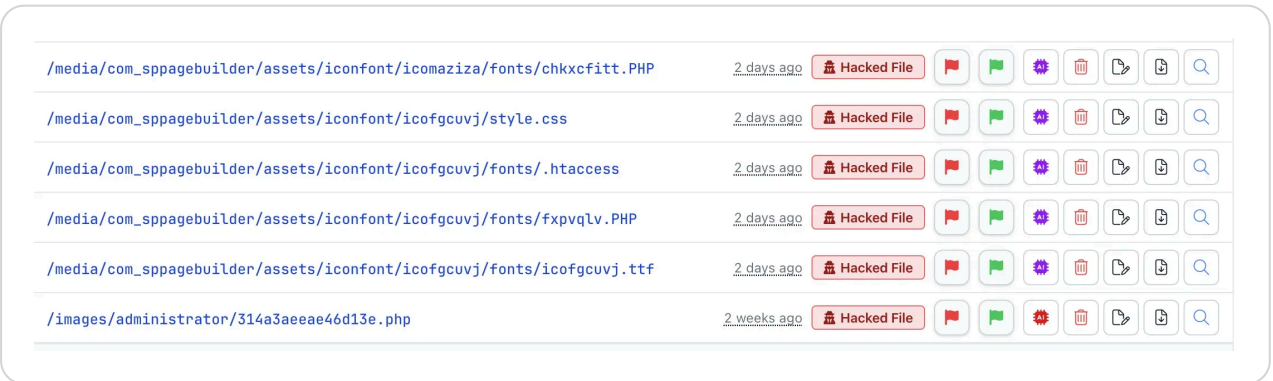
audit against the previous one and flags anything that changed in between. A backdoor that lay quiet through your last clean scan and woke up this week shows up here as a diff, in red, with the note “we highly recommend that you review these files.”

Two more checks in the same audit hunt the rest of the second-wave kit. **Check Files That Can Attempt To Upload Files** looks specifically for the dropper and uploader class of file, the small scripts an attacker leaves behind precisely so they can return and upload more later. **Changes To Core Joomla Files Should Be Avoided** (and its WordPress equivalent) hashes every core file against the official release, so injected JavaScript or a modified `index.php` is caught even when the file looks like it belongs.

> **Note**
>
> An audit is a point-in-time snapshot. For the gap between audits, mySites.guru also offers a near-real-time File Watch List: add files like configuration.php, wp-config.php or your template's index.php, and the on-site plugin recomputes their checksum on every page load and emails you the instant the content changes.

The reason this matters is that the audit reads every file on the server, line by line, including the dot-prefixed and oddly-named ones a scanner never opens. During the JCE wave, attackers dropped webshells with `.xml.php` double extensions into `tmp`, `images` and `media` folders. A scan of your rendered homepage will never find a `.xml.php` file sitting in `/images`. A file-level audit that hashes everything will. That is the whole difference between checking what your visitors see and checking what is actually on your server.

Here is what that looks like on a real site, caught in an audit. These are confirmed backdoors, flagged “Hacked File” against a known-bad hash, sitting quietly and waiting:



Look at the dates. One shell, `314a3aeeae46d13e.php` in `/images/administrator`, was planted two weeks ago. The rest, scattered through `/media/com_sppagebuilder/assets/iconfont/` as `.PHP` files and a `.htaccess` hidden in a fonts folder, landed two days ago. Two separate visits, two waves, both dormant. None of them changed a single thing a visitor would see, which is exactly why an external scanner that only reads your rendered pages walks straight past them. mySites.guru found all six because it audits under the hood: every file, every line.

What does a dormant dropper actually do?

A dropper is a small uploader script that does nothing visible when it is first planted. It waits. When the attacker comes back, it pulls down whatever they want next: a fuller webshell, an SEO spam directory, a credit-card skimmer, a crypto drainer. Uploaders are not a rare exotic. In Sucuri's analysis of website reinfection, the single most common backdoor type found was an uploader, "a PHP script that allows attackers to upload any file that they want." Attackers favour them precisely because they enable the later waves.

This is why the timeline of a compromise looks the way it does. The first wave is fast and quiet. It exploits the vulnerability, plants the dropper, maybe creates a hidden admin account, and leaves. There is often no defacement, no obvious redirect, nothing a casual look at the site would catch. Owners who do notice clean up the one file they found and assume it is over. It is not over. The dropper is still there, and so is whatever else the attacker seeded for persistence.

The WordPress supply-chain backdoor in the Essential Plugin portfolio is the cleanest illustration of the timeline. A buyer acquired the plugins, planted a backdoor in August 2025, and let it sit **dormant for eight months** before activating it in April 2026 to inject SEO spam and open a remote-code-execution endpoint. Eight months between the plant and the payload. A scan run in, say, December would have shown a perfectly healthy site.

Why does a cleaned site get hacked again within minutes?

Because the attacker built reinfection into the first wave on purpose. The visible malware is the part you are meant to find and remove. Underneath it sits the persistence layer: rogue admin accounts, compromised FTP credentials, and malicious cron jobs whose only job is to rewrite the payload the moment it disappears.

Sucuri documented a cron job that runs every single minute, checks whether the webshell is still present, and re-decodes it from base64 if it has gone, setting the file permissions to resist deletion. Their conclusion is worth quoting directly:

> **Warning**
> Even if you restore all your files and database from a clean backup and close all known vulnerabilities, the malicious scheduled tasks will still be there and will reinfect your site in a few minutes. Restoring files is not the same as closing every backdoor.

This is the norm, not a rare worst case. In Sucuri's 2023 hacked-website data, **49.21% of compromised sites contained at least one backdoor**, and among sites with database-level malware, **55.2% had a malicious admin user** left behind for re-entry. Layered persistence is how these campaigns are designed. Removing one backdoor when the site has three, plus a rogue admin and a cron job, just resets the clock.

The JCE wave is a textbook example of designing for repeat entry. Attackers did not leave a single way back in, they left a row of them. This audit found fourteen rogue editor profiles, all named `__pwn_traverse__`, each one configured to allow uploading dangerous `.php` and wildcard files through the editor, plus a machine-generated profile (`J144757`) matching the attacker's naming pattern. Every one of those profiles is a standing door. Delete the webshell and they walk back through the next profile, upload another, and carry on. That is how a site gets hacked again, and again, and again, long after the first cleanup looked like it worked.

162	__pwn_traverse__ (8)	Profile allows uploading dangerous files (.php / wildcard) through the editor
163	__pwn_traverse__ (9)	Profile allows uploading dangerous files (.php / wildcard) through the editor
164	__pwn_traverse__ (10)	Profile allows uploading dangerous files (.php / wildcard) through the editor
165	__pwn_traverse__ (11)	Profile allows uploading dangerous files (.php / wildcard) through the editor
166	__pwn_traverse__ (12)	Profile allows uploading dangerous files (.php / wildcard) through the editor
167	__pwn_traverse__ (13)	Profile allows uploading dangerous files (.php / wildcard) through the editor
168	__pwn_traverse__ (14)	Profile allows uploading dangerous files (.php / wildcard) through the editor
169	J144757	Machine-generated name matching the attacker label pattern (short prefix + long digit run)

The same redundancy shows up in the shell files themselves. On this site the attacker dropped the identical payload into `/images/media` five times over, changing only the case of the extension: `sauk872_1781619699.php`, `.pHp` and `.pHP`, alongside `loqnip.php` and `pnszqf.php`, all planted in the same visit two weeks ago. The casing trick is deliberate. A hand-written cleanup that searches for `*.php` in lowercase, or a server rule that blocks one casing, leaves the `.pHp` and `.pHP` copies live and executable. A hash-based audit does not care about the extension at all, so it flagged every copy as a confirmed hacked file.

The second wave on the page: injected JavaScript, redirects and spam

Once the dropper has done its work, the second-wave payload is built for money. The dominant forms are malicious JavaScript and conditional redirects. Injected JavaScript in a template or theme file runs in every visitor's browser. A conditional redirect

bounces traffic from Google to spam or phishing while leaving the site looking normal when you visit it directly. Sucuri's 2024 scan data found malware and malicious redirects on **74.7% of infected sites** and SEO spam on **38.4%**, alongside crypto drainers and credit-card skimmers for direct financial theft.

The mass `.htaccess` injection we built a one-click cleaner for is this exact mechanism at scale. On a hacked Joomla or WordPress site we have seen close to 9,000 malicious `.htaccess` files, one in nearly every folder, each carrying a redirect or PHP-injection rule. They are spread that wide for coverage and persistence: a hidden backdoor rewrites them within minutes of deletion, so cleaning the root file alone achieves nothing. That is the second wave in physical form, thousands of files planted to survive your cleanup, with a dropper standing by to replace any you remove.

The newer trend is worse for anyone relying on file scans alone. Campaigns like Sign1 increasingly store their payload in the database (in WordPress options rows) rather than in files, which is precisely why a once-and-done file scan misses them and continuous, multi-layer monitoring does not.

Why monitoring beats a one-time scan

Every reinfection report lands on the same point. Cleanup is necessary but it is not enough, because the attacker planned for your cleanup. What actually protects a site is continuous visibility: knowing what changed, and when, so a dropper that wakes up three weeks after the breach is caught the moment it touches a file.

Sucuri's own recommendation for self-reinstalling malware is "a file integrity monitoring service that resides mostly outside of the environment itself," comparing file hashes against known-good copies so you can see tampering even when the malware is built to hide. That is the principle mySites.guru runs on across every site you manage: an audit that diffs files between scans, an uploader hunter for the dropper class specifically, core-file hash verification, renamed-file detection for the `.bak` and `.old` stash trick, and a near-real-time File Watch List for the files you most want to guard. The hash list behind it carries more than 14,000 confirmed known-bad files and over

2,000 hand-written content patterns, updated daily, and a hack found on one connected site is checked for on every other site's next audit.

When we find an actively exploited Joomla zero-day, we do not just forward a CVE. With the iCagenda flaw we reproduced it ourselves, confirmed it was being exploited in the wild before any patch existed, filed responsible disclosure with the vendor, alerted every affected account, then code-reviewed JoomliC's 4.0.8 fix the same day it shipped and told users it was safe to reinstate. That is the level of attention the second wave demands, and it is why monitoring is not a slogan here, it is how the tooling is built.

If you suspect a site is compromised, start with whether your site is hacked and the full hacked-site cleanup process, then use the hack and backdoor detection to find what the first wave left behind. Start a free audit and see what is already sitting on your server, waiting.

Further Reading

- An Overview of Website Reinfection Vectors — Sucuri on why uploaders are the most common backdoor and how attackers guarantee they can return.
- Attackers Abuse Cron Jobs to Reinfect Websites — the mechanism behind sites that reinfect within minutes of cleanup.
- Web Shells — Sucuri on how webshells evade detection without continuous monitoring.
- RCE in the Joomla Content Editor Extension — YesWeHack's technical write-up of the JCE chain (CVE-2026-48907).
- CISA Known Exploited Vulnerabilities Catalog — the federal advisory that confirmed the JCE flaw was being actively exploited.

Frequently Asked Questions

Why does my site keep getting hacked after I clean it?

Because the first wave plants more than the visible payload. Attackers leave dormant droppers, hidden admin accounts and malicious cron jobs that re-create the malware after you remove it. Sucuri has documented cron jobs that rewrite a webshell every minute, so a cleanup that misses one backdoor reinfects within minutes.

What is a dormant dropper file?

A dropper is a small uploader script planted during the first hack that does nothing visible at first. It sits quietly until the attacker returns, then it pulls down fresh backdoors, webshells or spam payloads. Uploaders are the single most common backdoor type found on compromised sites precisely because they enable these later waves.

How do I detect a second-wave attack on Joomla or WordPress?

A browser-level scan of your homepage will not see it, because the dropper changes files on disk, not the rendered page. You need file-level detection: compare every file against the previous audit, flag anything modified between scans, and check the full file hashes against a list of known-bad files. mySites.guru's audit does all three.

Will restoring a clean backup fix a hacked site?

Not on its own. If the attacker planted a malicious cron job or compromised your FTP credentials, restoring files leaves those untouched and the site reinfects. Sucuri's guidance is blunt: even a full restore from clean backup will not help if the malicious scheduled tasks remain. You have to close every backdoor, not just replace the files.

What were the recent Joomla zero-days that triggered this wave?

Three unauthenticated remote-code-execution flaws landed in June 2026: JCE (CVE-2026-48907, added to CISA's Known Exploited Vulnerabilities catalog), SP Page Builder (CVE-2026-48908) and iCagenda (patched in 4.0.8). All three let an attacker with no login upload and run PHP, which is how the first-wave droppers got onto thousands of sites.

How does mySites.guru catch the second wave?

The audit includes a 'Files Modified Between Audits' check that diffs every file against your last scan, an uploader-detection check that hunts dropper code specifically, core-file integrity checking, and a near-real-time File Watch List that emails you the moment a

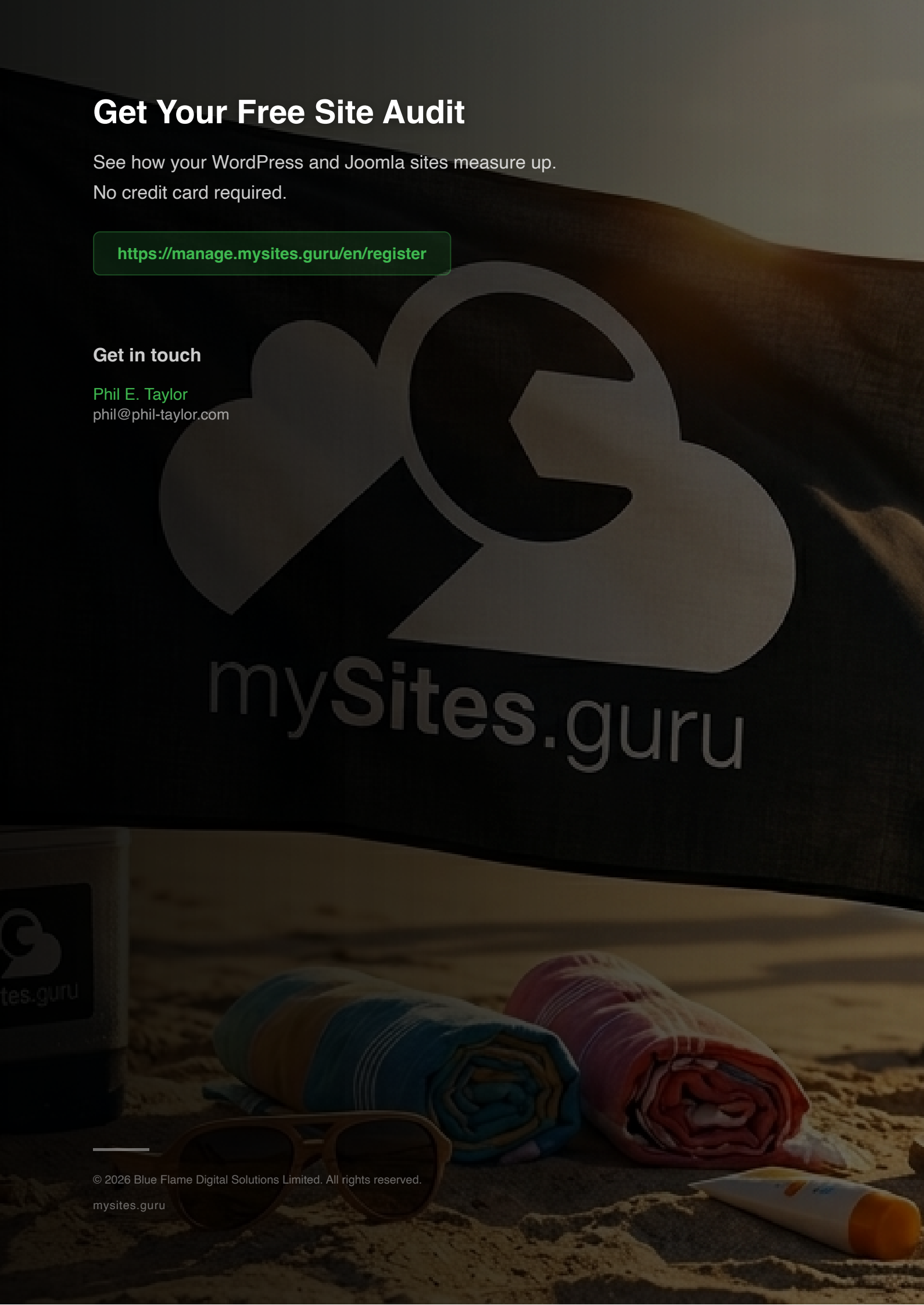
watched file's content changes. Together they surface the dormant files a one-time cleanup leaves behind.

Get Your Free Site Audit

See how your WordPress and Joomla sites measure up.

No credit card required.

https://manage.mysites.guru/en/register



Get in touch

Phil E. Taylor
phil@phil-taylor.com