Twelve Joomla Extension Vulnerabilities We Found and Disclosed in a Month (And more to come!)

In four weeks mySites.guru found and responsibly disclosed twelve security issues in popular Joomla extensions, most of them critical. Here is the full roundup and what to do about it.

Phil E. Taylor | 13 July 2026

Active Joomla Extension security alerts: Twelve vulnerabilities found this month • Helix3 • JCE • RSFiles • Balbooa Forms • SP Page Builder

Between mid-June and mid-July 2026, mySites.guru found and responsibly disclosed **twelve separate security vulnerabilities across twelve popular Joomla extensions**. Most were critical. Four rated the maximum CVSS 4.0 score of 10.0. Several were being exploited in the wild within hours of the fix going public. Every one was reported privately to the vendor first. Nine already have a vendor fix out; the last three are in active disclosure right now, with fixes pending, and we are holding the exploit detail and the names on those until they ship.

FOUND, REPORTED AND FIXED BY MYSITES.GURU

Every one of these was found, researched, privately disclosed, coordinated to a fixed release and written up by us, here at mySites.guru. This is what improving the Joomla ecosystem looks like from the inside: one report at a time. In the short term it is a headache for some Joomla admins, no question, but the whole ecosystem is in better shape for it. The people who run their sites through mySites.guru got the heads-up first wherever a coordinated timeline allowed, plus the exhaustive toolset to find and fix every affected site fast.

This is the roundup: what each flaw was, the version that fixes it, and the pattern connecting almost all of them. If you run Joomla, treat the table below as your update list.

A quick apology before you dive in: this post got waaay too long. We will be slicing it into shorter, single-point pieces in future posts, but we have a lot of information coming out soon and wanted to get it all down in one place first.

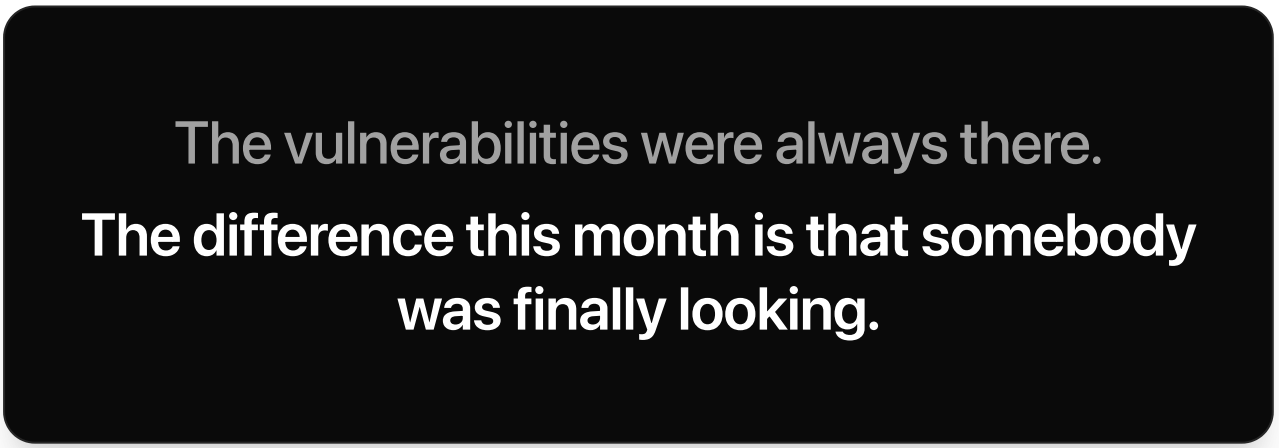
Why we went looking

This whole run started with an argument. A friend (Yes I have friends, despite what people think haha), in the aftermath of the JCE hack, put it to me that Joomla extensions were simply more secure than WordPress plugins. I told him that was the wrong conclusion. The gap is not security, it is attention.

The WordPress ecosystem pays people to find plugin bugs. There are bounty programs, Wordfence and others compensate researchers for reports, and there is even a black market that will buy a working plugin exploit. That money pulls a large, constant crowd of skilled eyes onto WordPress plugins. Joomla has almost none of that. The extensions are not audited less because they are safer, they are audited less because there is no reward for doing it. The bugs sit there undiscovered, which reads as “secure” right up until someone bothers to look.

So I bothered to look.

Reading the actual source of the extensions our customers run, I was finding **one to two critical or high-severity unauthenticated remote code execution flaws every single day, for over a week**. Not obscure edge cases. Anonymous, upload-a-PHP-file-and-run-it flaws, in some of the most widely deployed extensions in the ecosystem.



TL;DR

- It started as an argument: a friend claimed Joomla extensions were more secure than WordPress plugins. They are not, **nobody looks** because there is no bounty money in Joomla. So we looked

- Looking turned up **one to two critical or high unauthenticated RCE flaws every day for over a week**
- In about four weeks, mySites.guru found and responsibly disclosed **twelve Joomla extension vulnerabilities**
- Six were **remote code execution** through file upload (five unauthenticated, one authenticated), three were **unauthenticated SQL injection**, one an **unauthenticated write to stored XSS**, one an **unauthenticated file write and delete**, and one more, in an events and booking component, we are holding under wraps until it is patched
- **Four scored CVSS 10.0**: PageBuilder CK, SP Page Builder, Balbooa Forms and RSFiles!
- **Nine are already patched** - get onto the fixed versions in the table now. Three more are in active disclosure, redacted until their vendors ship fixes
- Almost all shared one root cause: a **public front-end endpoint that trusted anonymous input**
- mySites.guru flags every connected Joomla site still on a vulnerable version, automatically

Every vulnerability in this post was reported privately to its vendor first. We publish full detail after the patch, not before: for the nine already fixed you get the whole story, and for the three still in active disclosure we describe the risk but withhold the name and the exploit until a fix ships. The point is to get sites updated, never to hand attackers a recipe.

The vulnerabilities we found in a single month

Extension	Flaw	CVE	CVSS	Our write-up
Phoca Download	Authenticated upload to RCE	CVE-2026-57828	9.0	Read

Extension	Flaw	CVE	CVSS	Our write-up
RSFiles!	Unauthenticated upload to RCE	CVE-2026-57827	10.0	Read
AcyMailing	Unauthenticated SQL injection	CVE-2026-56292	8.7	Read
Balbooa Forms	Unauthenticated upload to RCE	CVE-2026-56291	10.0	Read
Helix Ultimate	Unauthenticated menu write to stored XSS	none assigned	High	Read
Helix3	Unauthenticated file write and delete	CVE-2026-49049	High	Read
PageBuilder CK	Unauthenticated upload to RCE	CVE-2026-56290	10.0	Read
SP Page Builder	Unauthenticated upload to RCE	CVE-2026-48908	10.0	Read
iCagenda	Unauthenticated upload to RCE	CVE-2026-48939	9.8	Read
Redacted (in disclosure)	Unauthenticated SQL injection	Pending	8.7	Coming soon
Redacted (in disclosure)	Unauthenticated SQL injection	Pending	8.7	Coming soon
Redacted (in disclosure)	Redacted	Pending	Redacted	Coming soon

The last three rows are still in active disclosure, and we have redacted their names on purpose: naming an unpatched flaw is just publishing an exploit. Two are unauthenticated SQL injections in different vendors' components, each handing an anonymous visitor a full read of the site database. The third is a flaw in a widely used events and booking component. One of the three will take a while to fix, so it may sit

redacted here for some time. We will update this post, and link each full write-up, the moment its vendor ships a fix.

NONE OF THESE IS A MINOR BUG

Every score in that table sits between **8.7 and 10.0**. These are not low-severity housekeeping issues, they are the most dangerous classes of web vulnerability there are: unauthenticated remote code execution, where an anonymous visitor runs their own code on your server, and unauthenticated full-database read, where an anonymous visitor walks off with every user account and password hash. It genuinely does not get much worse than this.

CISA put four of these on its must-patch-now list

When we say critical, we mean it, and now the U.S. government agrees.

The official seal of the U.S. Cybersecurity and Infrastructure Security Agency

In a single week, the U.S. Cybersecurity and Infrastructure Security Agency (CISA) added **four of the vulnerabilities in this roundup** to its Known Exploited Vulnerabilities catalog, the authoritative list of bugs CISA has confirmed are being exploited in the wild right now. A CVSS score is a severity opinion. A place on the KEV catalog is evidence: it means the attacks were observed happening, not theorised, and under Binding Operational Directive 26-04 every U.S. federal agency was legally ordered to patch each one within three days of it being listed.

CISA KNOWN EXPLOITED VULNERABILITIES CATALOG

Two CISA alerts, five vulnerabilities between them. Four were ours.

Confirmed exploited in the wild. A three-day federal deadline to patch every one.

CVE	Extension	Added to CISA KEV	Federal patch deadline
CVE-2026-56290	PageBuilder CK	7 July 2026	10 July 2026
CVE-2026-48908	SP Page Builder	7 July 2026	10 July 2026
CVE-2026-48939	iCagenda	10 July 2026	13 July 2026
CVE-2026-56291	Balbooa Forms	10 July 2026	13 July 2026

They came in two public alerts, both now permanent record. The “CISA Adds Three Known Exploited Vulnerabilities to Catalog” alert on 7 July was two of our Joomla findings plus one unrelated flaw. The “CISA Adds Two Known Exploited Vulnerabilities to Catalog” alert on 10 July was both ours.

Here is the honest detail, and it sharpens the point rather than blunting it. CISA’s catalog lists the CVE, the vendor and the product. It does not print the name of the person who found the bug, so you will not see “mySites.guru” on a government web page. Our name is on the CVE records instead, which is exactly where the credit for finding and reporting each one belongs. Read the two together and the sequence is not in doubt: we found these, we reported them privately, the vendors shipped fixes, and then a national cyber-defence agency independently confirmed they were already being exploited and made patching them compulsory across the entire U.S. federal government.

That is about as strong an outside verdict on security research as it gets. Not our marketing, not a CVSS calculator we ran ourselves, but CISA reaching the same conclusion we did and backing it with the force of law. When this post calls these bugs critical, that is what critical means in practice.

These are not fringe Joomla extensions

It would be comforting to write this off as a run of obscure, one-developer hobby extensions that nobody serious installs. It is the opposite. These are some of the most established, most trusted and most widely deployed names in the whole Joomla ecosystem: page builders and template frameworks that sit under a huge number of live sites, a newsletter platform that ships for WordPress too, mature form, calendar and document components that agencies roll out as standard furniture. Between them they run on an enormous number of production sites. The vendors are long-standing, respected members of the Joomla community, most with a genuinely good security track record.

That is the part worth sitting with. If extensions of this calibre, from teams who take security seriously, all carried a critical bug on the same public surface inside a single month, then “we only install popular, well-reviewed extensions” is not the safety net people treat it as. Popularity and a good reputation tell you the code is maintained and the developer answers a disclosure email quickly. They do not tell you the code is free of the one endpoint that trusts the wrong input. We are not talking about small players here, and that is exactly why it matters.

We went after the biggest names on purpose

There is an easy version of this exercise and a hard version, and we deliberately chose the hard one.

The easy version is to trawl the long tail: the abandoned, single-download, one-afternoon extensions that nobody has touched in years. That corner of any ecosystem, Joomla or WordPress, is riddled with holes. Finding a critical bug there proves nothing,

because nobody serious runs that software and breaking it helps no one. It is easy, and it is pointless.

We did the opposite. We pointed the research squarely at the extensions that matter to the people we protect: the ones at the top of the installed list across mySites.guru users' sites. The page builders, template frameworks, and the form, calendar, document and newsletter components that professional agencies deploy as standard, built and backed by some of the most respected, longest-established developers in the Joomla community. These are the names you would choose precisely because they are popular, maintained and trusted.

These extensions do not just run on reputable client sites. Several of the ones in this roundup are installed on the official `*.joomla.org` websites themselves. The Joomla Project runs this code on its own infrastructure, on the properties that represent Joomla to the world. There is no stronger vote of confidence, no clearer "trusted, vetted, safe to install" signal, than being chosen by the people who make Joomla, for Joomla's own sites. And it was still vulnerable.

We did not have to go digging in the gutter to fill a list. Every vulnerability in this post is in software a careful, security-conscious site owner would happily install, from a developer with a real reputation to protect. The critical bugs were in the flagship extensions, on the most reputable sites, the whole time.

Is Joomla insecure by default?

No, and it is worth stating plainly, because a post like this can leave the wrong impression. Not one of these vulnerabilities is in Joomla itself. Every single one is in a third-party **extension**: a component, plugin, module or template framework that someone chose to install on top of Joomla. Joomla core is not the weak point here, and nothing in this roundup says otherwise.

Joomla the platform has one of the stronger security stories in the CMS world: a dedicated Security Strike Team, its own CVE Numbering Authority (the Joomla CNA) that coordinates disclosures exactly like these, a signed and hardened core update

channel, and a long record of shipping fixes quickly. A default Joomla install, kept up to date, is a solid, well-defended piece of software.

The risk lives where it lives on every CMS: in the third-party ecosystem bolted on around the core. WordPress lives or dies by its plugins, and Joomla by its extensions. That is where code quality varies most, where independent review is thinnest, and where an attacker goes looking first. It is precisely why “keep Joomla updated” is only half the job. The other half is keeping every extension updated too, and knowing which ones are exposed at any given moment.

So the headline is not “Joomla is insecure”. It is that the extensions you add to Joomla are code too, and they deserve the same discipline you already apply to the core.

The critical file-upload RCEs in Joomla extensions

Six of the twelve were the same shape of bug: a front-end upload endpoint that accepted a file from an anonymous visitor without properly checking who sent it or what it was, letting an attacker drop a PHP file into a web-served folder and execute it.

- PageBuilder CK (CVE-2026-56290, CVSS 10.0). A front-end task gated only by a CSRF token, with no authentication or authorisation, accepted an upload and let the attacker choose the destination folder. Fixed in 3.6.0, with back-patches for the Joomla 3 and 4 branches. Exploited in the wild within a day.
- Balbooa Forms (CVE-2026-56291, CVSS 10.0). The form attachment upload took a file from anyone, with no login, no CSRF token and no type check, and trusted the supplied filename. A zero-day, confirmed exploited in the wild. Fixed in 2.4.1.
- RSFiles! (CVE-2026-57827, CVSS 10.0). The front-end upload task was reachable with no login and no token, and the write path did no file-type check, so anyone could drop a PHP file into the public downloads folder and run it. Fixed in 1.17.12.
- SP Page Builder (CVE-2026-48908, CVSS 10.0). An unauthenticated icon-upload endpoint gave remote code execution, and was being used to plant hidden Super User accounts. Fixed in 6.6.2.

- iCagenda (CVE-2026-48939, CVSS 9.8). The public “Submit an Event” form accepted uploads with no server-side type check, plus an access-control bypass that let guests create events. A zero-day, exploited in the wild. Fixed in 4.0.8 (and 3.9.15 for the legacy branch).
- Phoca Download (CVE-2026-57828, CVSS 9.0). This one needs a logged-in member and the non-default user-upload feature, so it is authenticated rather than anonymous, but the member-upload path skipped the file-type allow-list and let a member upload and run a PHP file. Fixed in 6.1.3.

The Joomla Extension SQL injections

Three of the twelve were SQL injection rather than file upload: not code execution, but a **read**. A public front-end endpoint placed request input into a database query with no sanitising, so an anonymous visitor could read any table in the database, including user accounts and password hashes. They score lower than the upload flaws (8.7 rather than 10.0) only because they read data rather than executing code, and every one of the three is the same public-endpoint-meets-unchecked-input pattern.

- AcyMailing (CVE-2026-56292, CVSS 8.7). A public front-end endpoint placed request input into a query unsanitised. It affects the WordPress plugin as well as the Joomla extension, because both are built from the same code. Fixed in 10.11.1 on both platforms.
- **Redacted, in coordinated disclosure** (CVE pending, CVSS 8.7). A widely installed Joomla component carries an unauthenticated SQL injection of the same class, with the same anonymous full-database-read outcome. This one has **no fix out yet, and it may take a while**, so we are withholding even the name until its vendor ships a patch. Worth noting: on shared hosting where the database account is over-privileged, a read like this can reach every database on the server, not just the one site.
- **Redacted, in coordinated disclosure** (CVE pending, CVSS 8.7). A second widely installed component, from a different vendor, carries the same class of unauthenticated SQL injection and the same full-database-read outcome. A fix is

> pending. We are deliberately not naming the extension or vendor yet, because naming an unpatched flaw is just publishing an exploit. We will update this post with the name, the CVE and the full write-up the moment it is patched.

Unauthenticated SQL injection often means complete database compromise

It is tempting to file a “read-only” SQL injection under low priority. Do not. Unauthenticated SQL injection is one of the most damaging classes of web vulnerability there is, and “read-only” badly undersells what an attacker walks away with.

Start with what a single anonymous request can read. Joomla’s `#__users` table holds every account, Super Users included, with their bcrypt password hashes. Those hashes crack offline at the attacker’s leisure, and any weak one that falls hands over a real admin login with no further exploit needed. The `#__session` table can expose live sessions. Extension parameters in `#__extensions` routinely hold API keys and integration credentials. Every article, every custom field, every piece of data the application keeps in the database is readable. One injection, and the confidentiality of the whole application is gone.

The reach often does not stop at one site, either. On a great deal of shared and reseller hosting the database account the site connects with is over-privileged, sometimes able to see every schema on the MySQL server. Where that is true, a single injection in one Joomla site can read every other database on the box, other customers’ sites included, and in the worst setups the server’s own `mysql.user` table. The blast radius is not “this site’s data”, it is “everything this database login can see”.

“Blind” shrinks none of this, it only slows it down. A blind injection returns no data in the page, so tooling asks the database one true-or-false question at a time and rebuilds the answer bit by bit. That is a throughput detail for the attacker, not a safety margin for you: automated tools walk a full table out in seconds to minutes, and the end state is identical to a noisy injection.

And a read is frequently just the opening move. Depending on the database privileges and the exact injection point, the same flaw can escalate to writing rows (planting a Super User, flipping a user into an admin group, injecting a stored-XSS payload into content), or, where the database login holds the `FILE` privilege, reading server files with `LOAD_FILE` and writing a web shell with `INTO OUTFILE`, turning a “read” into remote code execution. Even when it stays a pure read, “the attacker now has every credential and secret the site stores” is a full compromise in every way that counts.

That is why we treat the three SQL injections in this roundup as high-severity emergencies rather than paperwork, and why the answer is never “add a firewall rule and move on”. A WAF can blunt the delivery. Only the patch closes the hole.

The Helix template framework issues

Two of the findings were in JoomShaper’s template frameworks, which sit on an enormous number of Joomla sites.

- Helix3 (CVE-2026-49049) had an unauthenticated file write, arbitrary file delete and template-parameter overwrite reachable through `com_ajax` before any token or permission check. The template-parameter angle matters: it is being used to inject a defacement and wallet-drainer payload straight into the database, where file scanners never look. Fixed in 3.1.1, and it shipped under a two-word “Security Update” changelog that badly understated it.
- Helix Ultimate had several `com_ajax` actions running with no CSRF or permission check, enabling an unauthenticated menu-settings write that leads to stored cross-site scripting, plus an in-folder file delete, an open redirect and an unprotected template export. Fixed in 2.2.7. No CVE has been assigned.

Why these Joomla vulnerabilities nearly all look the same

Read the list again and the same phrase keeps appearing: a public front-end endpoint. That is not a coincidence, it is the whole story.

Extensions have to expose endpoints that anonymous visitors reach without logging in. A contact or booking form has to accept a submission. A calendar has to serve its events. A file manager has to hand out downloads. A page builder has to load and save layout fragments over AJAX. None of that can sit behind a login, because the public is the intended audience. Joomla's `com_ajax` gateway and front-end component tasks exist precisely to expose this surface.

Every one of those public endpoints is attack surface. The bug is almost never the endpoint existing. It is the endpoint **trusting** what arrives: taking a filename the visitor supplied and writing it to disk, taking a request parameter and dropping it into a database query, reaching a file-write or delete without first checking a token and a permission. An upload handler with no type allow-list turns "submit an event" into "run my code". A query built by string concatenation turns "search the newsletter" into "read the password table". This is the public-endpoint-meets-unchecked-input pattern, and it is behind ten of these twelve findings. Once you have seen it a few times, you know exactly where to look in the next extension.

None of this is new: authentication, authorisation and Joomla security 101

It is worth being blunt about how ordinary these bugs are. Nothing in this roundup is novel. There is no exotic new attack class, no clever chain nobody had thought of, no zero-day technique. Every single one comes down to one of the oldest lessons in web application security, and they split cleanly into the three questions every request should have to answer:

JOOMLA SECURITY 101 — AUTHN / AUTHZ / INPUT

Three questions every request must answer

01 AUTHENTICATION SKIPPED

Who is making this request?

A `com_ajax` action that fires before any token or login check has skipped this entirely.

02 **AUTHORISATION** SKIPPED

Are they allowed to do it?

A front-end task that reaches a file write, a delete or an upload without checking a permission has skipped this one.

03 **INPUT VALIDATION** SKIPPED

Can this input be trusted?

An upload with no file-type allow-list, or a query built by gluing request text together, has skipped this one.

That is it. That is the whole of it, twelve times over. These three ideas are the first week of any secure-development course and the top of every checklist OWASP has published for twenty years. They are not obscure corners that only bite under exotic conditions, they are the fundamentals, and experienced developers are expected to get them right by default and without thinking about it. This is not written to shame anyone, because it is genuinely the reassuring part: the defences are well understood, cheap and boring. The fix is never the hard part. The failure is almost always the same human one, a single endpoint that, on a busy day, skipped a check that everybody already knows to make.

Five things the Joomla Project could do to protect every site

None of this is Joomla core's fault, as we said. But the platform is uniquely placed to make this entire class of bug far harder for an extension developer to write in the first place. Five changes would make it much harder to ship, and the first two alone would have stopped every vulnerability in this post before it shipped.

1. Ship a tiny SQL-injection filter as a core plugin

There is essentially never a legitimate reason for a raw, SQL-looking payload to arrive as a request parameter on a public `index.php` endpoint. A default-on system plugin that inspects incoming request values and rejects the obvious injection shapes (union selects, stacked queries, comment sequences, `information_schema` probing) would neutralise the delivery of every SQL injection in this roundup. It does not need to be clever or exhaustive. It could genuinely be ten lines. Call it a mini-WAF.

Aside, because we enjoyed this rather too much: "Miniwaf" is also a popular crunchy rolled-wafer snack made by the Hai Ha Confectionery Company in Vietnam. Ours has fewer calories and blocks considerably more attacks.

There is precedent for exactly this, too. In 2019 the Joomla Project adopted the TYPO3 phar stream wrapper across the entire CMS, a single central guard that shut down `phar://` object-injection attacks so no individual extension had to defend against them alone. It was only deprecated in 5.3 and removed in 6.0 once PHP 8 fixed the underlying issue and made it redundant. A default-on request filter is the very same move: one small piece of core code that protects every extension at once.

2. Make that same mini-WAF refuse front-end file uploads by default

The other half of this roundup is anonymous file upload. A core plugin that simply blocks multipart file uploads to public front-end endpoints unless the receiving component has explicitly opted in would have stopped the upload RCEs cold. Most of the vulnerable endpoints never intended to accept a file from an anonymous visitor at all. A default-deny posture, opt-in per component, turns “the developer forgot to check” into “safe unless someone deliberately allowed it”.

Those two small features, a request filter and an upload gate, would between them have blocked every single vulnerability in this post.

3. Provide one hardened, official file-upload API

Today every extension author rolls their own upload handling, and every roll is a fresh chance to get it wrong. Joomla should offer a single, battle-tested, unit-tested upload

service that developers call instead of writing their own, with the safe behaviour baked in and not optional:

- A strict, non-bypassable extension **allow-list** (permit `jpg`, `png`, `pdf`; never a deny-list that forgets `phtml`, `phar`, `php7`)
- Real **content inspection**, verifying the file's actual magic bytes and MIME, not its name or the client-supplied `Content-Type`
- **Polyglot detection**, rejecting files that are valid as two formats at once (the classic image or PDF header that is also runnable PHP), which sail straight past a naive magic-byte check
- **Double-extension and null-byte** handling (`shell.php.jpg`, `shell.php%00.jpg`)
- **Filename normalisation**, generating a random server-side name rather than trusting the uploaded one
- Writing to a location that is **not web-served**, or is hardened against execution, by default
- Enforced **size and count limits**, plus per-user rate limiting
- Clear, prominent **documentation**, so the safe path is the obvious path a developer reaches for

None of this is exotic. It is the boring, known-good checklist that every one of these extensions had to implement by hand, and that at least six of them got wrong in a single month. Build it once, test it to death, and the whole ecosystem gets safer by default.

4. Stop pretending a complicated .htaccess is security

Joomla ships a long, intimidating `.htaccess.txt` full of rewrite and blocking rules, and a cottage industry of "hardening" guides tells site owners to pile on more. It looks reassuring. It is mostly theatre. Every vulnerability in this roundup arrived through an approved, normal request to `index.php`, the one path every `.htaccess` is built to allow, so those rules never got a look in. A file complex enough that almost nobody who copies it understands it, and that does not stop the attacks that actually happen, is not security. It is the feeling of security, plus a support burden. The platform should stop

leaning on it as a headline defence and put the protection where the requests actually arrive: in the code, in a request filter, in a safe upload API. Damage-control rules in an upload directory are worth having, but they are the seatbelt, not the brakes, and they should be presented that way.

5. Give AJAX endpoints one simple, standard way to handle auth and authz

Almost every extension that exposes a `com_ajax` handler or a front-end task writes its own authentication and authorisation from scratch: checking the CSRF token here, remembering to call `authorise()` there, deciding for itself whether a guest should be let through. Most of the bugs in this post are that exact decision going wrong, or being skipped, on a single endpoint. When every developer hand-rolls the same gate, sooner or later one of them leaves the latch off.

Joomla should give AJAX and front-end endpoints one declarative way to say “this action needs a valid token” and “this action needs this permission”, enforced by the framework before the handler ever runs. Better still, make the safe state the default: an endpoint that declares nothing gets a token check and a login requirement automatically, and a developer has to consciously, visibly opt a task out to expose it to the public. Getting authentication and authorisation right should be one line of declaration, not a discipline every extension author has to remember to apply perfectly, every single time.

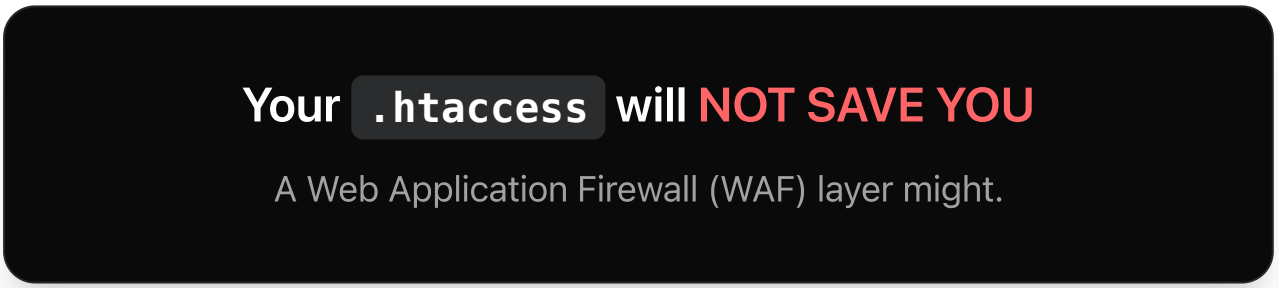
We are not the Joomla Project - actually we are physically banned from contributing to the Joomla project, like other major players in the Joomla ecosystem (but yet they love to protect their toxic inner circle - go figure) and we do not get to set its roadmap. But we read a great deal of extension code, and from where we sit these five changes would remove most of the ground this month’s vulnerabilities grew in.

Why .htaccess is damage control, not a Joomla security fix

A common reaction to a run of upload vulnerabilities is “I have got an `.htaccess`, I am fine”. You are not, and it is worth being precise about why.

These flaws were reached through **approved, normal endpoints**. The malicious request looked like a legitimate form submission or AJAX call, so it sailed through routing and rewrite rules. An `.htaccess` that only blocks odd-looking URLs never sees a problem, because nothing about the request was odd. The application accepted it on purpose.

Where `.htaccess` genuinely helps is **after** the file has been written: a rule in your upload directories that refuses to execute PHP downgrades a remote code execution into an arbitrary file write. That is a real reduction in blast radius, but it is not safety. An attacker who can still write files can fill your disk and take the site down, host illegal content under your domain, or leave a payload waiting for the day execution gets re-enabled. And none of it touches the SQL injection class at all: there is no file to block, the attacker reads your database straight out of a normal query.



This all deserves its own full treatment, and it is getting one. We have a dedicated post coming soon on exactly why an `.htaccess` protects far less than most site owners think, which requests ride straight through it, and the hardening (and the WAF layer) that actually earns its place. The short version for today: an `.htaccess` is a seatbelt. Worth having, capable of turning a catastrophe into a bad day, and no reason whatsoever to skip the actual fix. Update the extension.

More Joomla disclosures are already in the pipeline

This roundup covers what we can talk about. It is not the whole of what we have found. Three of the twelve above are still inside their disclosure windows, redacted until their

vendors ship fixes: two unauthenticated SQL injections in different vendors' code, and a flaw in a widely used events and booking component. Beyond those, we are still holding further confirmed vulnerabilities, reported or about to be reported, and not yet patched. We will not name the extensions or vendors until a fix exists, for the obvious reason that naming an unpatched flaw is just publishing an exploit. Described only by what an attacker gains:

- A Joomla component where an anonymous request can enumerate the name and email address of every registered user on the site, plus a related file-handling weakness on the same public surface.

And that is only what is already confirmed. **We are still researching another 10 potential critical or high-severity issues in well known Joomla extensions.** Some will turn out to be nothing. On the last month's strike rate, most will not.

We are also aiming higher than extensions. Together with another security researcher, and with AI assistance, we are working to chain several individually low-severity issues into a single working compromise of Joomla core itself, on the current 6.1.2 release. It is early days, but our honest assessment is that a remote compromise of Joomla 6.1.2 is likely to be provable, and we expect to be able to demonstrate it. If we get there, it follows the exact same path as everything above: reported privately to the Joomla Security Strike Team first, with no public detail until a fix exists.

Each of these will get its own full write-up here the moment its vendor ships a fix, exactly as the ones above did. If you are a mySites.guru subscriber, you do not have to track any of this by hand: the affected sites get flagged automatically as soon as each vulnerability is public and in our database.

AI is shrinking the gap between a Joomla patch and mass exploitation

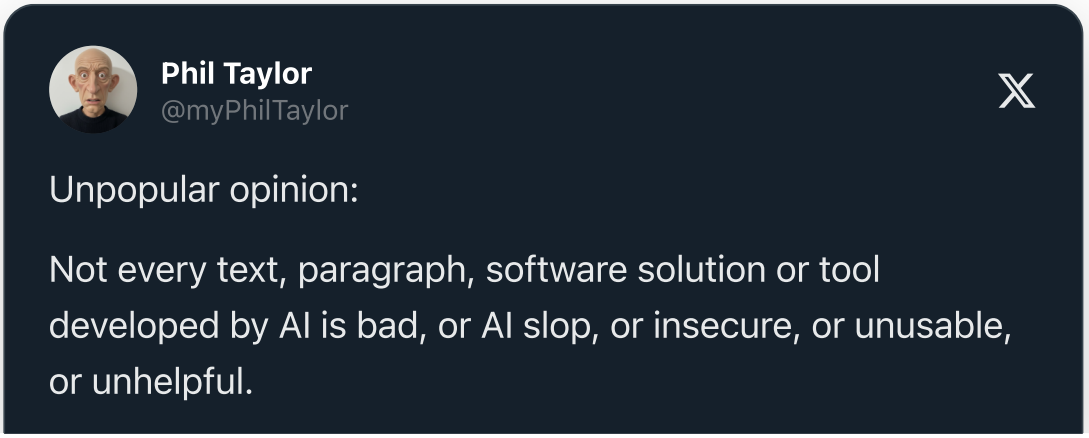
Here is the uncomfortable half of every disclosure. The moment a fix ships, the fix itself becomes a map. Anyone can diff the patched code against the previous version, see exactly which check was added and therefore which check was missing, and

reconstruct the exploit from that difference. This has always been true. What has changed is the speed.

An attacker can now feed a patch diff to an AI model, get a working understanding of the bug in minutes, generate a proof-of-concept, and turn automated scanning loose to find candidate sites running the vulnerable version, all in the time it used to take to read the changelog. Several of the flaws above were exploited in the wild within hours of the fix appearing. The window between “patched” and “hunted at scale” is now measured in hours, not weeks. That is the real reason “I will update at the weekend” is a losing strategy.

It is worth being straight about our own side of that, because the honest version is more reassuring than the marketing version. **Every one of these twelve issues was found by a human first**, by reading the actual extension source code and recognising a dangerous pattern, not by pointing a model at a repository and waiting. That human judgement was then backed by tooling we have built and sharpened over the last decade: the connector that inventories every extension and version across a portfolio of sites, the vulnerability matcher, the proof-of-concept harnesses, the local test rigs. AI came in only as a sprinkle at the end, a way to make sure no code path or edge case was missed once the human had already found the door. Human-led, tool-assisted, AI for completeness. The attackers are automating the weaponisation of your patches. The least you can do is not give them a week’s head start.

None of which means AI is the villain of the piece. The same tool cuts both ways, and how useful or how dangerous it turns out to be is mostly a statement about the person holding it.



A bad workman always blames his tools.

View on X

Better found by us than by an attacker

If you run one of these extensions and this post made your stomach drop, hold onto one thing: the bug existed whether or not we found it. The only real question was who would find it first, and what they would do next.

When we find one, a fixed sequence follows. A private report to the vendor. A fix built and shipped. A coordinated release. Only then a public write-up, with the exploit detail held back. The site owner gets a patch before they get the bad news, and nobody is handed a working exploit.

When someone with worse intentions finds the same bug, none of that happens. There is no email to the vendor, no patch, no warning. The first you hear of it is a defaced homepage, a wallet-drainer injected into every page, a mailbox full of spam sent from your server, or a quiet backdoor that sits unnoticed for months.

Same bug, opposite outcome. So yes, a run of disclosures like this is uncomfortable reading, but the uncomfortable version is the good one. The alternative was never "the bug never existed". The alternative is that someone found it and chose not to tell you. Better us.

How mySites.guru finds affected Joomla sites for you

Updating one site is easy. The hard part, when you manage dozens or hundreds of Joomla sites, is knowing **which** ones run PageBuilder CK, or RSFiles!, or AcyMailing, and which of those are still on a vulnerable version. Twelve disclosures in a month is twelve rounds of that question, and checking each admin panel by hand does not scale.

mySites.guru already tracks the installed extensions and versions on every connected site. As each of these vulnerabilities was disclosed, we added its affected range to our extension vulnerability database, so every connected site running a vulnerable version is flagged automatically. You see, in one place, exactly which sites need which update, and you can push the fix from the dashboard rather than logging into each one. You can even enable auto-updates for any Joomla extension so a fix like these is in place before an attacker gets there, and mass-update across every affected site from one screen.

This is what we built the platform for. You should not learn about a Joomla security issue from the news and then go audit a portfolio of client sites by hand: we did exactly that when we found every site running a vulnerable copy of JCE, and we repeat it for every disclosure here. The dashboard tells you what is exposed, lets you drop straight into any site's admin with one click to check or fix it, and does it all from one place.

What to do right now

- Work down the table above and get every affected extension onto its fixed version. Start with the CVSS 10.0 three: PageBuilder CK, Balbooa Forms and RSFiles!
- For the three still in active disclosure (all redacted until their vendors ship fixes), there is no patch to apply yet. Make sure a SQL-filtering firewall sits in front of any site running them, and update the instant their fixes ship. We will update this post the moment they do
- Take a backup before each update, and snapshot everything first if you manage more than a couple of sites
- If you manage many sites, let mySites.guru show you which are still exposed rather than checking by hand, and push the updates in bulk
- Harden your upload and download directories to refuse PHP execution, as damage control, not as a substitute for updating
- If you were on a vulnerable version of an upload flaw for any length of time, assume you may have been hit: find any hacked files and backdoors and follow our guide to fixing a hacked Joomla site

- If a SQL-injection-affected extension was ever live on your site, audit your Joomla database security and rotate anything an attacker could have read, starting with user password hashes
- Not on it yet? Sign up to mySites.guru, connect your Joomla and WordPress sites, and let the dashboard flag every one running a vulnerable extension for you, then update them in bulk. You can start with a free audit
- Subscribe to our announcement list so alerts like these reach you the moment they are public, instead of finding out from the news or, worse, from your logs

Further Reading

- Why public AJAX endpoints are a CMS security blind spot - the pattern behind almost all of these
- Our AcyMailing SQL injection disclosure - the same class of bug we found across several extensions this month
- Joomla! Security Centre and the Joomla CNA
- CISA Known Exploited Vulnerabilities catalog - four of these are on it, added in a single week
- OWASP: Unrestricted File Upload and SQL Injection
- How mySites.guru mass-updates extensions across every site
- Find every site running a vulnerable extension

Frequently Asked Questions

How many Joomla vulnerabilities did mySites.guru disclose in the last month?

Twelve, across twelve different Joomla extensions, between mid-June and mid-July 2026. Six were remote code execution through file upload (five unauthenticated, one authenticated), three were unauthenticated SQL injection, one was an unauthenticated write leading to stored cross-site scripting, one was an unauthenticated file write and delete, and one more, in an events and booking component, we are keeping under wraps until it is patched. Every one was reported privately to the vendor first. Nine already have a vendor fix out; three are still in active disclosure with fixes pending, and we have redacted their names until then.

Are all of these Joomla vulnerabilities patched?

Nine of the twelve are, and we withheld public detail on each until its fix was available. Three are still in active disclosure and we have left them unnamed until each vendor ships a fix: two unauthenticated SQL injections in different vendors' components, and a flaw in a widely used events and booking component. For those three we describe the risk but withhold the name and the exploit, and we will update this post the moment each is patched. For everything already fixed, the single most important action is to get each affected extension onto the fixed version in the table. Several were exploited in the wild within hours of the fix appearing, so do not treat any of them as low priority.

Why did mySites.guru go looking for Joomla vulnerabilities?

A friend argued that Joomla extensions were more secure than WordPress plugins after the JCE hack. We disagreed: the difference is not security, it is attention. The WordPress ecosystem pays researchers to hunt plugin bugs, through bounty programs like Wordfence and, less pleasantly, a black market that buys plugin exploits, so far more eyes are on WordPress plugins. Almost nobody looks at Joomla extensions with the same intensity. So we went looking, and found one to two critical or high-severity unauthenticated remote code execution flaws every day for over a week. The bugs were always there. Nobody was looking.

Which of these Joomla vulnerabilities was the most serious?

Four rated the maximum CVSS 4.0 score of 10.0: PageBuilder CK, SP Page Builder, Balbooa Forms and RSFiles!, all unauthenticated remote code execution through file upload. An anonymous visitor could upload and run a PHP file with no login. Phoca Download is also code execution but scores 9.0 because it needs a logged-in member. The AcyMailing SQL

injection was read-only, so it scores lower at 8.7, but it still exposes every user record and password hash.

Are any of these vulnerabilities in CISA's Known Exploited Vulnerabilities catalog?

Yes. The U.S. Cybersecurity and Infrastructure Security Agency added four of them to its Known Exploited Vulnerabilities (KEV) catalog in a single week: PageBuilder CK (CVE-2026-56290) and SP Page Builder (CVE-2026-48908) on 7 July 2026, then iCagenda (CVE-2026-48939) and Balbooa Forms (CVE-2026-56291) on 10 July 2026. A place on the KEV catalog means CISA has confirmed the flaw is being exploited in the wild, and under Binding Operational Directive 26-04 every U.S. federal agency was ordered to patch each one within three days. All four were found and disclosed by mySites.guru.

Does a firewall or an .htaccess file protect me from these?

Only partially, and never as a substitute for updating. A SQL-aware web application firewall blocks the AcyMailing-style injection, and a hardened .htaccess can stop an uploaded PHP file from executing, which limits the damage of the upload flaws. Neither stops an attacker filling your disk or dropping non-PHP files, and neither fixes the underlying bug. Update the extension.

How do I check which of my Joomla sites are affected?

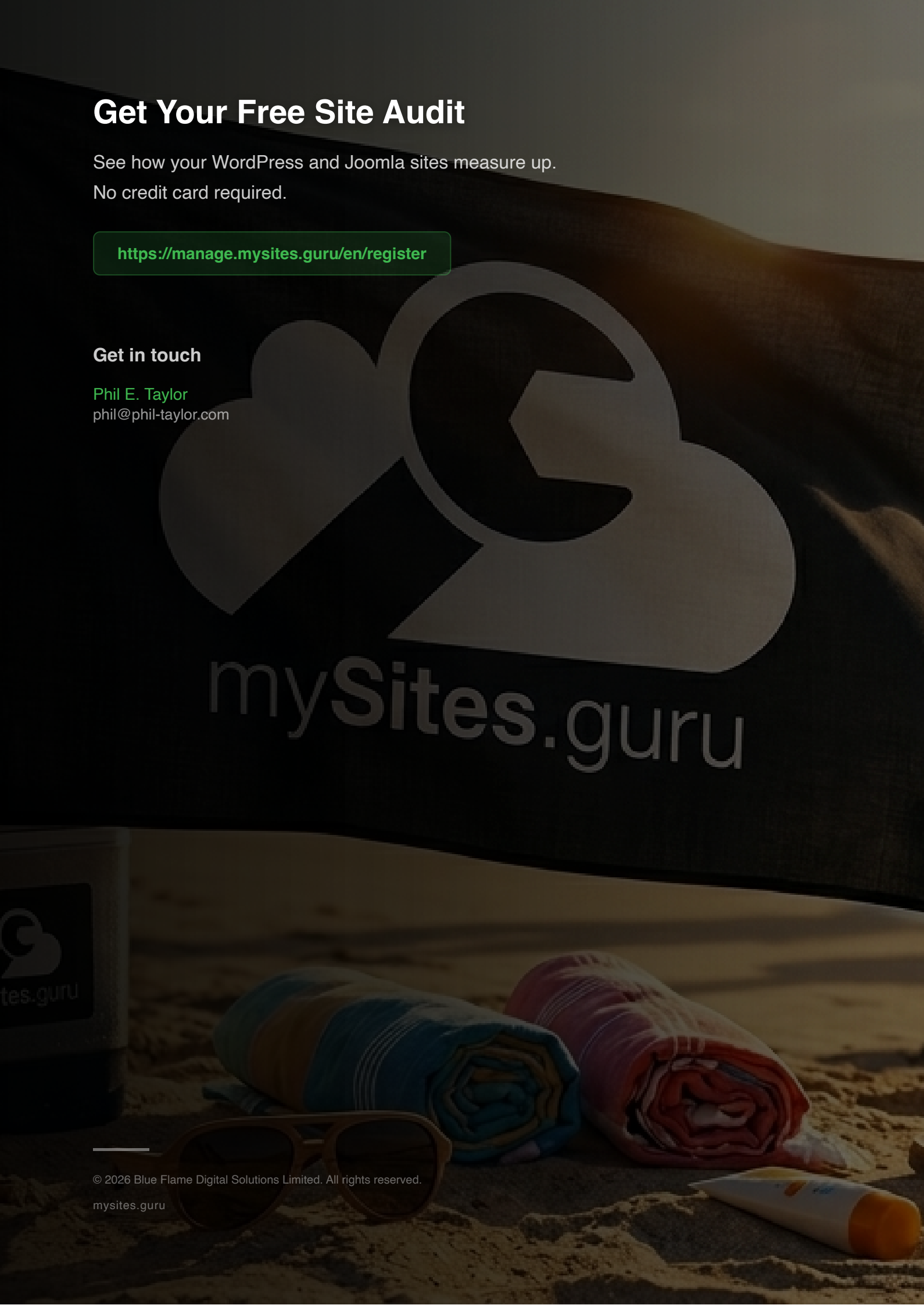
mySites.guru tracks the installed extensions and versions on every connected Joomla site, and we add each of these vulnerabilities to our extension vulnerability database as it is disclosed. Every connected site running an affected version is flagged automatically, so you see which sites need updating in one place instead of logging into each admin panel.

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