mySites.guru

A CVSS 10.0 Unauthenticated Upload in YOOtheme ZOO, Fixed in 4.1.64

YOOtheme ZOO (com_zoo) up to 4.1.63 had an unauthenticated file upload RCE scored CVSS 10.0, plus an unauthenticated SQL injection. All three flaws are fixed in 4.1.64.

Phil E. Taylor | 19 August 2026

ZOO is YOOtheme's content construction kit for Joomla, installed as the `com_zoo` component, and it has been in service long enough to be running on sites first built a decade ago. Every version up to and including 4.1.63 carries three vulnerabilities that need no login, no account, and no help from anyone on the site. The worst of them is scored **CVSS 10.0**, the maximum, and ends in an attacker running their own code on your server.

We found all three, reported them, and YOOtheme fixed them in **ZOO 4.1.64**, released on 19 August 2026. Update to 4.1.64 or later now.

TL;DR

TL;DR: YOOtheme ZOO (`com_zoo`) up to and including 4.1.63 has three unauthenticated flaws, all fixed in **4.1.64** on 19 August 2026, all found and reported by mySites.guru and all proved on a live test install rather than inferred from source. CVE-2026-74803 (**CVSS 10.0**, CWE-434) is an arbitrary file upload to remote code execution: the Image element on a front-end submission form checks the `Content-Type` the client sent and nothing else, so an anonymous visitor uploads a `.php` file declared as an image and it executes from inside the web root. CVE-2026-74804 (**CVSS 9.3**, CWE-89) is an unauthenticated SQL injection in the item element endpoint that needs no submission form at all, only an installed ZOO application, and returns whatever the database holds. CVE-2026-75114 (CVSS 5.1, CWE-601) is an open redirect. **Update every ZOO install to 4.1.64 or later**, then check `images/zoo/uploads/` for anything that is not an image.

What Was Wrong in YOOtheme ZOO 4.1.63?

Three separate things, in three separate parts of the component, with one property in common: none of them requires an account on the site.

Flaw	Identifier	Score	Needs a login?
Arbitrary file upload leading to remote code execution	CVE-2026-74803	**10.0 Critical**	No
SQL injection in the item element endpoint	CVE-2026-74804	**9.3 Critical**	No
Open redirect in the Twitter comment callback	CVE-2026-75114	5.1 Medium	No

All three are credited to Phil Taylor of mySites.guru in the CVE records published by the Joomla project, which acts as the CVE Numbering Authority for Joomla extensions. The affected range on every record is 1.0.0 to 4.1.63, so this is not a bug introduced in a recent build. It has been there a long time.

> **If you run ZOO 3.x, read this bit**
>
> The CVE records list the affected range as 1.0.0 to 4.1.63, which includes the entire ZOO 3.x line. YOOtheme shipped a fix on the 4.x line only. There is no 3.x security release. If you are on 3.x you are inside the affected range with nowhere to update to, so the realistic options are moving the site to the 4.x line or taking ZOO off it. Leaving a 3.x install published and reachable is not one of them.

The Image Element Trusted the Browser's Word

This is the CVSS 10.0 one, and the mechanism is worth understanding because the same mistake turns up across the whole Joomla and WordPress extension ecosystem.

ZOO lets visitors submit content through a front-end submission form. That is a documented, intended feature, and guest submissions are the default rather than an

exotic configuration. If the form includes an Image element, which the stock layouts do, visitors can attach a picture.

The Image element validates that attachment. It builds a validator configured with a MIME type group of `image` and a maximum size, which reads, at a glance, like exactly the right check. The problem is which value that validator inspects. It reads the `Content-Type` header the client attached to the upload.

That header is not a property of the file. It is a label chosen by whoever is sending the request, and there is nothing stopping them writing whatever they like in it. So the check amounts to asking the attacker what kind of file they are uploading and believing the answer.

Nothing else closes the gap:

- the file's actual contents are never inspected, so the bytes can be anything at all;
- no allow-list of permitted extensions is configured for this element;
- the filename goes through Joomla's `File::makeSafe()`, which strips dangerous characters but happily preserves a `.php` extension;
- the file is then written into `images/zoo/uploads/`, a directory inside the web root, where the web server executes PHP.

Chain those together and an anonymous visitor uploads a PHP file, tells ZOO it is a JPEG, and then requests it in a browser to run it. That is unauthenticated remote code execution, the most severe outcome a web vulnerability can have, and it is why the Joomla CNA scored it at the top of the scale with high impact across confidentiality, integrity and availability, both for the component and for the wider system.

We proved it end to end on a Joomla 5.4.7 test install: a benign file that printed the server's own `php_uname()` output was uploaded as an unauthenticated guest and returned that output over HTTP. We are not publishing the request.

Why "Check the MIME Type" Keeps Failing

There are two completely different things people mean by validating a file's type, and only one of them is a security control.

The first is reading the `Content-Type` header sent with the upload. That is a hint supplied by the client. It is useful for deciding how to display something, and useless for deciding whether to trust it, because the client is the attacker.

The second is examining the file's own contents server-side, checking the magic bytes, and refusing anything that does not match an allow-list of extensions you have decided are acceptable. That one is a security control.

They read almost identically in code review. A line that says "check this is in the image MIME group" looks responsible either way, and the difference is one variable deep. That is exactly why this class of bug survives audits, and it is the same underlying mistake we wrote up in Page Builder CK, Balbooa Forms, iCagenda and RSFiles over the past few months.

The rule that actually holds: **never let the extension of a stored file be decided by anything the caller sent you.** Generate the name and the extension yourself from a fixed list, and the entire bug class stops existing.

How Bad Is the ZOO SQL Injection?

Bad enough to matter on its own, and it has a wider reach than the upload flaw because it has no preconditions.

CVE-2026-74804 sits in `ItemController::element()`. Two values taken straight from the request, `filter_type` and the `type_filter` array, are pasted into the SQL condition as string comparisons with no quoting and no escaping of any kind. The condition string is then concatenated into the query verbatim by the table layer underneath.

An anonymous visitor can therefore close the string early and write their own SQL. In testing that let us do two things. First, bypass the filters the query exists to enforce, so items that were unpublished and items restricted to logged-in users came back to a

guest alongside the public ones. Second, attach a `UNION` and read arbitrary data: we pulled back the MySQL version and the database name as an unauthenticated request to prove the reach without touching customer data.

The practical difference from the upload flaw is the precondition. The upload needs a published submission form containing an Image element that guests can reach. The SQL injection needs nothing except an installed ZOO application, which every deployed ZOO site has by definition. **A site with no submission form at all is still fully exposed to this one.**

That matters for triage. If you were planning to skip the update on ZOO sites that do not take submissions, do not.

The Open Redirect Is the Least of It

CVE-2026-75114 is the small one, and in a release with a CVSS 10.0 in it there is a temptation to ignore the 5.1 entirely. It is still worth patching.

`CommentController::twitterAuthenticate()` takes a `referer` parameter from the request and hands it straight to `setRedirect()` without checking the scheme or the host. Ask for it and the site issues an HTTP 303 to wherever you named. It works whether or not Twitter authentication has ever been configured on the site.

Nobody loses a server to an open redirect. What they lose is the reputation of their domain, because a link that genuinely starts at your trusted site and silently ends somewhere else is exactly what a phishing campaign wants, and it survives the “check the link before you click it” advice that most staff training is built on.

How Do I Find Every ZOO Install Across My Joomla Sites?

If you run one Joomla site, log into it, open the extension manager, and read the ZOO version off the screen. That takes a minute and you are done.

If you run thirty, or two hundred, that approach is where the update quietly stops happening. You get through the first dozen, something urgent arrives, and the rest of the list never gets checked. The sites that end up compromised are almost never the ones nobody knew about. They are the ones on a list somebody meant to get back to.

mySites.guru keeps an inventory of every extension on every connected Joomla and WordPress site. Search it for ZOO and you get every install, with its version, on one screen, and you can push the update to all of them from there.

The vulnerability rules go a step further and flag the versions rather than waiting for you to compare numbers yourself. The rule for this one went live on the day the CVEs were published, so any connected site running ZOO below 4.1.64 is already marked. Our own numbers across connected sites when that rule went in were not encouraging: **hundreds of sites still on a vulnerable ZOO, and not a single one yet on 4.1.64**, which is what you would expect on release day and exactly why the next fortnight is the dangerous part. A meaningful share of those were still on the 3.x line, which has no fix to move to at all.

What to Do Right Now

In order, most urgent first.

1. **Update ZOO to 4.1.64 or later on every site.** This closes all three flaws. It is the only step that fixes anything; everything below is about checking what may already have happened.
2. **If a site is on ZOO 3.x**, there is no security release for that branch. Plan the move to 4.x, and in the meantime treat the site as exposed rather than patched.
3. **Look in `images/zoo/uploads/` for anything that is not an image.** Anything ending in `.php` there should be treated as a web shell until proven otherwise. Widen the search to any unexpected PHP file anywhere under `images/`.
4. **Check your Joomla administrator accounts** for any you do not recognise, and review recently modified files across the site.

5. **If the site was internet-facing on an old version, treat stored password hashes as read.** The SQL injection could reach the user table. Force a password reset and rotate the Joomla secret in `configuration.php`.
6. **Do not skip sites that have no submission form.** They are not exposed to the upload flaw, but they are fully exposed to the SQL injection, which needs nothing but an installed ZOO application.

If any of that turns up something you would rather not deal with yourself, fix.mySites.guru is the paid service where we do it for you: the site gets patched, checked for backdoors, and handed back clean.

How Do I Tell If a ZOO Site Was Already Hit?

Updating stops the next attempt. It tells you nothing about the last one, and on a flaw this old the honest answer is that the window has been open for a long time.

Start with the upload folder, because that is where the evidence would be. `images/zoo/uploads/` should contain images and nothing else. A `.php` file there, or a file with a double extension, or an image with a modification date that does not match anything you submitted, is worth treating as an incident rather than a curiosity.

Then look at the things a successful upload leads to. Administrator accounts you did not create. Scheduled tasks nobody set up. PHP files with recent modification times in directories that should be static. Outbound connections from the web server that you cannot account for.

The reason to do this rather than assume you are fine: an anonymous file upload is not a flaw that needs a targeted attacker. Mass scanners work through the Joomla extension list continuously, and we have watched several of these upload flaws get probed at scale within days of the details reaching the public, which the Page Builder CK case documented in detail. mySites.guru runs those checks, including the suspect content tool, across every connected site automatically.

Why Front-End Submission Forms Keep Producing This Bug

Step back from ZOO for a moment, because it is not really about ZOO.

Look at the pattern in what we have published this year: Page Builder CK, Balbooa Forms, iCagenda, RSFiles, and now ZOO. Different vendors, different codebases, different decades in some cases. The same bug.

What they share is a feature requirement that pulls directly against the security requirement. The whole point of a front-end submission form is that a stranger can use it without an account, and if it accepts pictures then it must accept a file from that stranger. Every safeguard you would reach for first is one the attacker controls: the filename, the extension in the filename, and the `Content-Type` label. The only inputs that are not attacker-controlled are the file's actual bytes and the rules you wrote yourself, and checking those is more work than checking a header.

So the code that looks careful gets written, review sees a MIME check and moves on, and the bug sits there for years. ZOO's affected range starts at 1.0.0.

If you build Joomla or WordPress extensions, the takeaway is short: **the file's declared type is not evidence.** Read the bytes, enforce an extension allow-list you control, generate the stored filename yourself, and if you can, put the upload directory somewhere the web server will never execute anything.

If you run sites rather than build them, the takeaway is shorter still. Any extension that takes uploads from the public is a standing risk, and the number of them installed across your sites is the thing to keep an eye on. Our vulnerable extension list tracks the ones we know about.

Further Reading

- CVE-2026-74803, CVE-2026-74804 and CVE-2026-75114 at CVE.org, the authoritative records

- YOOtheme's ZOO changelog
- CWE-434: Unrestricted Upload of File with Dangerous Type at MITRE
- CWE-89: SQL Injection at MITRE
- The Joomla Security Strike Team's vulnerable extensions list
- Our standard for disclosing Joomla extension vulnerabilities
- A month of Joomla security disclosures, the wider pattern these keep fitting

Frequently Asked Questions

What are the security flaws in YOOtheme ZOO?

Three, all reachable without logging in, all fixed in ZOO 4.1.64. CVE-2026-74803 (CVSS 10.0) is an arbitrary file upload leading to remote code execution: the Image element on a front-end submission form decides whether an upload is an image by reading the Content-Type header the browser sent, which the person sending the request controls completely. Send a PHP file labelled image/jpeg and it is written into a public folder and runs. CVE-2026-74804 (CVSS 9.3) is a SQL injection in the item element endpoint, where two request values are pasted into the query with no quoting, allowing an anonymous visitor to read the database. CVE-2026-75114 (CVSS 5.1) is an open redirect in the Twitter comment callback.

Which version of ZOO is affected, and which one fixes it?

Every ZOO release up to and including 4.1.63 is affected. The CVE records give the affected range as 1.0.0 to 4.1.63 on all three flaws. YOOtheme fixed all of them in ZOO 4.1.64, released on 19 August 2026, the same day the CVEs were published. Update to 4.1.64 or later. If you run ZOO 3.x, you are inside the affected range but there is no 3.x fix release, so you need to move to the 4.x line or take the extension off the site.

Do I have to have a submission form for the upload flaw to affect me?

For the CVSS 10.0 upload flaw, yes. It runs through the Image element on a published ZOO front-end submission form that guests can reach, which is ZOO's default configuration for submissions rather than an unusual setup. The SQL injection has no such precondition: it needs nothing but an installed ZOO application, which every deployed ZOO site has. So a site with no submission form is still exposed to an anonymous full database read, and should still update.

How do I find every ZOO install across the Joomla sites I manage?

Manually you would log in to every Joomla site in turn and read its installed extensions list. With mySites.guru you open the extension inventory, search for ZOO, and see every connected site running it alongside its installed version on one screen. Anything below 4.1.64 needs the update, and you can push it to all of them from the same place. mySites.guru also flags the vulnerable versions automatically against its Joomla vulnerability rules.

My site runs an old ZOO. How do I know if it was already attacked?

Updating closes the door but does not tell you whether anyone already walked through it. Look in the ZOO upload folder, by default images/zoo/uploads/, for anything that is not an image, particularly files ending in .php. Widen that to any unexpected PHP file anywhere under images/. Check your Joomla user list for administrator accounts you do not recognise, and look at recently modified files across the site. Because the SQL injection could read the user table, treat stored password hashes as exposed and force a password reset if the site was on an old version and internet-facing.

Is ZOO still maintained?

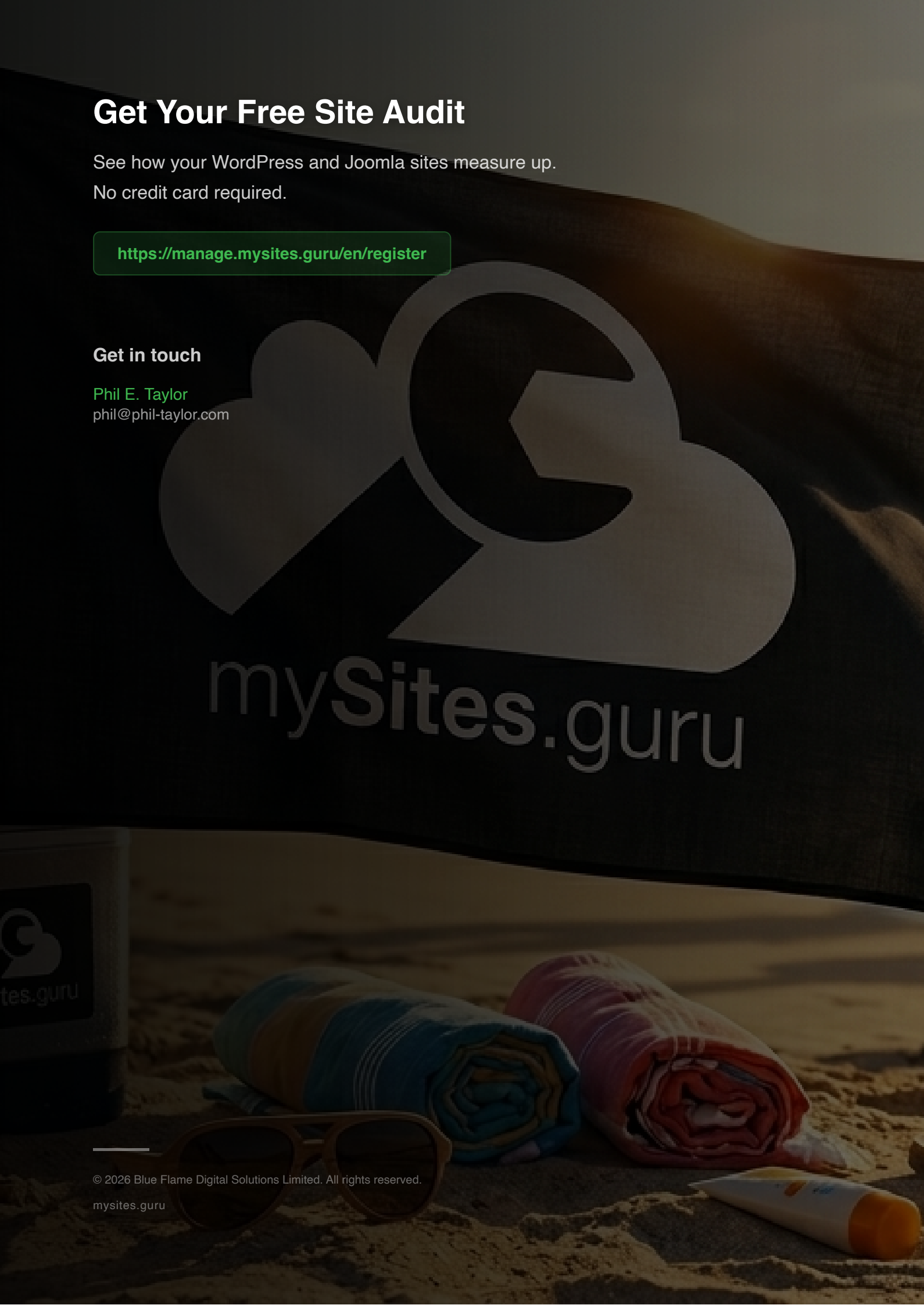
Yes. ZOO is YOOtheme's long-running content construction kit for Joomla, and 4.1.64 shipped on the same day the CVEs were published, which is a fast turnaround by any standard. YOOtheme also described each fix openly in the changelog rather than burying it in general maintenance wording, announced the release publicly, and credited the researcher who reported the flaws by name. That is better handling than several Joomla extension vendors have managed this year.

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