mySites.guru

Unauthenticated SQL Injection in Quix Page Builder found by mySites.guru

mySites.guru found and reported an unauthenticated SQL injection in Quix Page Builder for Joomla. An anonymous request to a front-end element endpoint could read the whole site database. Fixed in Quix 6.2.1, update now.

Phil E. Taylor | 15 July 2026

⚠ **Active Joomla Extension security alerts:** Thirteen vulnerabilities found this month • DPCalendar • JCE • RSFiles • Quix • SP Page Builder

Quix is one of the most widely installed drag-and-drop page builders for Joomla, the tool a lot of agencies and site owners reach for when they want to lay out a page without touching code. During routine security research on the extensions our customers rely on, **mySites.guru discovered an unauthenticated SQL injection vulnerability in Quix Page Builder, and reported it to ThemeXpert before disclosing anything publicly.** The fix is now available: Quix **6.2.1**.

If you run Quix on any Joomla site, update to 6.2.1 now. If you manage more than a handful of sites, read on for how to find every affected one at once.

TL;DR

- mySites.guru found an **unauthenticated SQL injection** in Quix Page Builder and reported it to ThemeXpert
- The flaw is in one of Quix's front-end element endpoints, present in **6.2.0 and every earlier version**; fixed in **6.2.1**
- The endpoint took an **article id from an anonymous request and placed it straight into a database query** without turning it into a number, so an **anonymous visitor could read data from any table**: user accounts, password hashes, the Joomla secret, and anything else in the database, from draft content and API keys to personal data and ecommerce orders
- It is the **error-based** kind of SQL injection, not the blind kind. The endpoint reflects the database error back in its response, so the data comes straight out, one value per request, with no guessing
- **Update to Quix 6.2.1 immediately**
- We score it **CVSS 4.0 8.7 (High)**. It is a read-only injection, which is the only reason it is not Critical

- A CVE has been **requested through the Joomla CNA**, crediting Phil Taylor of mySites.guru; the identifier is pending
- The injection was **one of six security issues we reported**, all six fixed in 6.2.1, along with hardening of the bundled JMedia media manager. One update closes the lot
- Credit to ThemeXpert: they shipped the fix **the same day we reported it**
- mySites.guru already flags every connected Joomla site still running a vulnerable version, so you do not have to check each site by hand

> mySites.guru discovered this security issue and reported it to ThemeXpert before publishing details. We withheld the exact request and any proof-of-concept until a fix was available and site owners had a reasonable window to update. This is how we handle every vulnerability we find: fix first, publish second.

What is the vulnerability?

Quix builds pages out of elements, and some of those elements load their content over a front-end AJAX endpoint that anonymous visitors can reach without logging in. One of them, the element that pulls in a single Joomla article, took an article `id` from the incoming request and used it to check whether that article exists before rendering. That existence check placed the id into a database query **without casting it to an integer first**, the textbook shape of a SQL injection.

The id does not arrive as a bare number. It is carried inside an encoded request parameter, which the endpoint decodes and reads the id back out of. Decoding it does nothing to make it safe: the value that comes out is still fully attacker-controlled, and it reaches the query unchanged. Joomla's standard input filtering, which constrained only the outer encoding, never looks at the decoded value. Quotes, parentheses and SQL keywords all survive.

So an attacker could supply a crafted id that turned the existence check into a query reading from any table in the Joomla database. No account needed, no CSRF token, nothing stolen first. It is the same public-endpoint-meets-unchecked-input pattern behind a long run of recent Joomla extension vulnerabilities.

In practice that means an unauthenticated attacker could read:

- Joomla user accounts, including usernames and password hashes
- The Joomla configuration secret and other stored data
- Article, content and extension data across the whole database

We are deliberately not publishing the exact request or a working proof-of-concept. The point of this post is to get people updated, not to hand attackers a recipe.

This one is not blind: the database talks back

There are two broad flavours of SQL injection, and which one you are dealing with changes how quickly an attacker gets the data out. In the quiet kind, a **blind** injection, the response never prints the data. The attacker asks the database true-or-false questions one at a time and reads the answer from how the page changes. It works, but it is slow, a character at a time down a binary search.

This Quix flaw is the other kind, the loud one: an **error-based** injection. The affected endpoint catches the database error and reflects the message straight back in its response. That hands an attacker a much shorter path. Instead of inferring a value bit by bit, they craft an id that forces the database to fail while embedding the data they want inside the error message, and the endpoint prints that message back to them. The requested value, a username, a password hash, the site secret, comes back in the response to a single request.

Do not read “error-based” as a technicality. It is worse for the defender than blind, not better. Blind injection at least makes an attacker work for each value. Error-based removes even that friction: the database does the talking, the endpoint does the printing, and the attacker reads the answer off the screen. The end state is the same as

any other full SQL injection, the entire contents of every table an anonymous request can reach, but it arrives faster and with less noise in your logs.

> Because this is error-based, there is no "it would take a long time to exploit" comfort to fall back on. The data returns in the response. Confidentiality is either intact or it is gone, and against an unpatched Quix it is gone in one request per value. Treat any site that ran a vulnerable version as one where the database could have been read.

What could a hacker actually do with this?

Download your entire database. Every table, every row, every column, from an anonymous request that looks exactly like ordinary traffic in your access log.

That is the honest answer, and we are not speculating about it. We reproduced complete database extraction against this flaw repeatedly on our own Quix test installations, pulling table after table out through the endpoint until there was nothing left to take. Separately, we confirmed vulnerable versions of Quix running in the wild on sites you would expect to know better: organisations with security teams, compliance obligations, and budgets, all of them one anonymous request away from handing over the lot. We did not touch their data, and we did not need to. The version number told us everything.

Here is what walking off with a Joomla database actually gets someone:

- **Every user account and password hash.** Hashes get cracked offline, at leisure, on hardware built for it. Any user who reused their password has handed over their other accounts too. A cracked Super User hash is simply an admin login, and no exploit is needed for the second visit.
- **The Joomla secret.** It signs sessions and tokens. With it, an attacker stops needing your password.

- **Everything your site was trusted with.** Customer records, order histories, addresses, private article drafts, form submissions, stored API keys for whatever else you connected. If it is regulated data, you now have a notifiable breach on your hands rather than a maintenance ticket.

The part that catches people out is how quiet it is. A database read writes nothing. No file changes, no new admin user, no web shell, no malware for a scanner to find, nothing in your integrity check. Your site looks perfect, because it is: the attacker took a copy and left the original exactly where it was.

This is why "restore a backup" is not a fix for a database read. Restoring undoes changes, and nothing changed. Once the data is copied it stays copied, and the only remaining moves are rotating every credential in it and telling the people whose data it was. Updating stops the next copy, not the one already taken.

You never needed AI to weaponise SQL injection

Every time a SQL injection is disclosed now, someone frames AI as the thing that made it dangerous. It is worth saying plainly that it did not. Fully automated SQL injection, error-based and blind alike, has been free, off-the-shelf tooling for about twenty years. sqlmap, the standard open-source tool, has automated error-based, boolean-based and time-based extraction and dumped entire databases since July 2006. Point-and-click extractors put the same power in front of absolute beginners even earlier. The underlying techniques were documented publicly in the early 2000s.

An injection a human would find tedious is a single sqlmap command that has been public since before a lot of today's extension code was even written. AI did not make SQL injection practical. sqlmap did that two decades ago. This matters because it kills the most common excuse for not updating: "nobody would really bother extracting a whole database by hand". Nobody does it by hand. They point a tool at it and walk away.

Joomla knows this class of bug intimately, and not only from its extensions. Its own core has shipped serious SQL injection flaws, and they did real damage long before anyone

had an AI assistant:

- **October 2015, `com_contenthistory` (CVE-2015-7857 and related).** An unauthenticated SQL injection in Joomla core, in every release from 3.2.0 to 3.4.4, let an anonymous attacker steal a live Super User session and take the whole site over. It was error-based, exactly like this Quix flaw, and it was being exploited in the wild within 24 hours of the fix, against a component that shipped enabled by default on millions of sites.
- **May 2017, `com_fields` (CVE-2017-8917).** An unauthenticated SQL injection in Joomla 3.7.0, through an ORDER BY parameter that public exploits drove both blind and error-based, exposing session tokens and password hashes. It was weaponised with public exploit code within days.

None of those needed AI. SQL injection has been fully compromising Joomla sites, core and extensions alike, for well over a decade, with tools any teenager could download for free. The stakes were never raised by AI. They were always this high, and the only reliable answer has always been the same one: update the affected code.

Sibling elements were hardened, this one was missed

This is the detail that makes the flaw interesting rather than embarrassing. ThemeXpert had already been through Quix's element code, hardening exactly this class of input, and had shipped that work one day earlier as Quix 6.2.0, a release whose changelog opens by announcing a completed security audit of the extension. The elements that load lists of articles cast their ids to integers and lean on Joomla's own parameterised article model. A related element that reads an article field got a strict allow-list of column names, with a comment in the code describing the fix. The team clearly knew the shape of the risk and had been closing it element by element.

The single-article existence check was the one that slipped through. It kept its hand-written pre-check, and that pre-check kept the raw id. It is the classic pattern of a hardening pass that fixes the neighbours and overlooks the helper next door. It is a reminder that "we fixed that class of bug" and "we fixed every instance of that class of

bug" are different statements, and the gap between them is where findings like this live. Our audit of that freshly hardened 6.2.0 build found six issues it had missed, this injection among them, and ThemeXpert fixed all six the following day in 6.2.1.

Read that as an argument for external eyes rather than a knock on ThemeXpert. An audit by the people who wrote the code is bounded by the same assumptions that produced the code, which is precisely why a second pass by someone else keeps finding things. 6.2.0 was a real and substantial piece of security work. It was also, inevitably, incomplete, and the only way anyone learns that is by looking.

The fix is a single line: cast the id to an integer before it reaches the query, so a crafted value can no longer change the SQL. That is exactly what Quix 6.2.1 does.

How serious is it?

Serious enough that we treated it as a priority disclosure. The thing that decides real-world impact here is short:

Whether the site has a web application firewall that filters SQL. Because the payload is literally SQL, a WAF often recognises and blocks it before it reaches Quix. Sites running bare, with no such protection, are the exposed ones.

Note what is *not* on that list. Unlike some front-end injection flaws, this one does not need the site to have any particular content published, or a specific menu item, or a visible page using the affected element. If Quix is installed, the endpoint is reachable, and the injection works. There is no second precondition for an attacker to satisfy. That makes the exposure broader than flaws that depend on the site being in a particular state.

> This is the case for defence in depth. A SQL-aware firewall in front of a vulnerable extension can catch this class of attack before it reaches the code, buying you time to update. It does not mean you can skip updating. Treat the firewall as the seatbelt and the patch as not crashing the car: you want both.

One more thing worth saying plainly: Quix is not obscure code on neglected sites. It is a paid, actively maintained page builder chosen precisely because it is polished and well supported, and it runs on plenty of professionally managed Joomla installations, including ThemeXpert's own. Being well-regarded is not the same as being safe. "We only use trusted, well-maintained extensions" is a good instinct, and on its own it is not a defence against a flaw in one of them. The most buttoned-up site in the world is still exposed if it is a version behind on the extension that has the hole.

Would a long, strong .htaccess have protected me against this?

Almost certainly not. A big hardened `.htaccess` is the security blanket a lot of Joomla owners reach for, and it guards a different kind of attack than this one. The exploit here is an ordinary, allowed request to `index.php`, the exact request your `.htaccess` exists to let through so the site works at all. The dangerous part is not the URL, it is the SQL hidden inside one request parameter, and a stock `.htaccess` never looks there.

To block this at the `.htaccess` layer you would need explicit rules that pattern-match SQL syntax in the request and return a 403. Hardly anyone runs those, they are brittle, and they tend to block real visitors as often as attackers. Inspecting request input for SQL is really the job of a web application firewall with SQL injection filtering switched on. Even then it only helps if that filtering is actually enabled: a WAF with its SQLi rules turned off, or left in log-only mode, waves this through as happily as no WAF at all. Protection you have to remember to configure is not the same as code that is not vulnerable in the first place.

The short version applies here directly. Update Quix.

Which versions are affected?

Quix **6.2.0**, the current release at the time of disclosure, and **every earlier version** are affected. The vulnerable pre-check helper is long-standing and predates the hardening pass that fixed its sibling elements, so there is no clean lower bound: the whole Quix

history below the fix is in range. The fix is in **6.2.1**, which casts the article id to an integer before it reaches the query so a crafted value can no longer change the SQL.

> If you run any version of Quix Page Builder earlier than 6.2.1, assume your site is affected and update now. Do not wait to confirm exploitation: because this is error-based, by the time you can confirm it, the data is already gone.

The SQL injection was not the only thing we reported

This post covers the SQL injection because it is the one an anonymous visitor can reach across the internet with a single request, and the one with the clearest consequence. It was one of six security issues we reported to ThemeXpert, and Quix 6.2.1 fixes all six. In the vendor's own words, the release also:

- Fixed a code-execution path via the Raw HTML element, so PHP tags in element content are now neutralised
- Fixed a path traversal and arbitrary file read in the Form element submission handler
- Fixed stored cross-site scripting through the icon field, including inline SVG
- Added an explicit permission check to the admin media manager endpoint
- Stopped reflecting internal error details in front-end AJAX responses

That last one is worth pausing on, because it is the mechanism this whole post rests on. The reflected database error is what made the injection error-based instead of blind, handing an attacker the data in the response rather than making them tease it out. Quix 6.2.1 closes the injection and stops the endpoint reflecting internal errors, so the same class of mistake elsewhere in the component has lost its loudspeaker.

ThemeXpert also shipped a hardened version of the bundled JMedia media manager (1.6.1) alongside it, which blocks executable and polyglot uploads and remote-URL SSRF, sanitises uploaded SVG, and serves files with hardened headers to close a media-picker XSS. If you run JMedia, take that update too.

We are not publishing detail on the other five for the same reason we are not publishing the SQL injection request: the fix is out, and the useful thing now is that you install it. One update closes all six.

Credit where it is due

We gave ThemeXpert a private report and withheld all public detail while they worked. They never needed the window. Quix 6.2.1 shipped the same day we reported, with the id cast to an integer and the injection closed, and ThemeXpert credited mySites.guru for the find.

That is the right response, and worth saying out loud. Plenty of vendors sit on a report for weeks. A same-day turnaround on six separate security findings, from a private report to a shipped release, is at the far end of good. Our thanks to the ThemeXpert team, and to their CTO Abu Huraira Bin Aman, who confirmed each fix back to us directly.

ThemeXpert is not the exception this month either. Most of the developers we have responsibly disclosed to during this run have responded the same way: grateful, quick to thank us, and shipping a fix promptly. The right thing for site owners to do in return is simple: apply the update they shipped.

SQL injection is a recurring theme right now

This Quix flaw is not an isolated find. It is one of several unauthenticated SQL injections we have uncovered in widely used Joomla extensions in a single research run, and they all share one root cause: a public front-end endpoint that takes anonymous request input and hands it to a database query without turning it into a safe value first.

- DPCalendar (CVE-2026-57831, CVSS 8.7). The same class of unauthenticated SQL injection in one of the most popular calendar and events extensions for Joomla. Fixed in 10.11.2 and 8.19.4.

- AcyMailing (CVE-2026-56292, CVSS 8.7). The same shape again in one of the most popular newsletter extensions for Joomla, and because it shares a codebase, the WordPress plugin too.
- EDocman. Another unauthenticated SQL injection from the same run, this time in a document-management component.
- The month of Joomla disclosures. Quix is one entry in a larger run of vulnerabilities we found and reported across popular Joomla extensions in a single month, several of them this exact SQL injection shape.
- Why AJAX endpoints are a CMS security blind spot. The underlying pattern, and why front-end element endpoints like Quix's keep producing this bug.

The lesson is not "avoid page builders". The reputable ones are well maintained, and this vendor fixed the issue when told. The lesson is that any extension exposing a front-end endpoint to anonymous visitors is attack surface, and keeping those extensions updated is security-critical, not just feature maintenance.

How do you update Quix safely?

1. **Take a backup first.** Before any extension update on a production Joomla site, back up the database and files. If you use mySites.guru, trigger a snapshot or a full backup so you have a clean starting point.
2. **Update through Joomla's Extensions manager, or in bulk from mySites.guru.** On a single site, open the Joomla administrator, go to System, then Update, then Extensions, and let Joomla pull Quix 6.2.1. If it does not appear, use Find Updates, or download the latest package from your ThemeXpert account and install it over the top. If you manage more than one site, use the mySites.guru mass update feature to upgrade Quix across every affected site from a single dashboard. You can even enable auto-updates for any Joomla extension so the next fix like this reaches your sites without you lifting a finger.
3. **Confirm the version.** After updating, open Components, then Quix, and check the version shown is 6.2.1 or later.

4. **Clear caches.** Clear Joomla's cache and any CDN or page cache so stale front-end assets do not linger.

Updating closes the door. It does not undo any data an attacker may already have read, so if you handle sensitive data and were on a vulnerable version for a long time, treat credentials and any exposed data as potentially known.

How do I find every Quix site I manage?

The first question after any extension security release is the awkward one: which of my sites actually run this? Up to about ten sites, you can log in to each Joomla admin and check the installed extensions list. Past that, you need a single view.

mySites.guru keeps a live inventory of every extension, template and framework on every Joomla and WordPress site in your account. You search for Quix once and get back every connected site running it, the version each one is on, and whether an update is available. No logging into forty admin panels one at a time.

Once you know which sites need it, the mass updater handles the rollout: tick the sites on an old version, push the update to all of them from one screen.

Why page builders are a quiet attack surface

This is a pattern, not a one-off, and it is why the class of bug matters more than this single instance. Page builders make attractive targets for the same reason they are useful: they render their content through front-end endpoints that anonymous visitors are meant to reach. An element that loads an article, a form that accepts a submission, a widget that fetches data, none of that can require a login, because the whole point is that visitors see it.

Every one of those public endpoints is attack surface. When the code behind them trusts request input, or hands it to a database query without escaping or casting, an anonymous visitor becomes an anonymous attacker. We have seen the same class of issue across page builders like PageBuilder CK, form builders like Balbooa Forms, template frameworks like Novarain, calendar tools like DPCalendar, and newsletter systems like AcyMailing: a public task that reaches a dangerous operation with too little checking in between.

The lesson for site owners is not "avoid page builders". They are fine, and the reputable ones are well maintained. The lesson is that **keeping them updated is security-critical, not just feature maintenance**. An extension two versions behind is not a cosmetic problem.

Stay Ahead of the Next One

This is one extension on one day. There will be another, because Joomla runs on thousands of third-party extensions and the ones that accept input from anonymous visitors keep producing bugs like this. The hard part is never the update itself. It is knowing a fix exists, knowing which of your sites are affected, and getting to them before an attacker does, across every extension on every site you look after.

That is the job mySites.guru does for you. It keeps a live inventory of every extension on every Joomla and WordPress site in your account, flags the ones with a known vulnerability, and lets you push the update to all of them from one screen. When

something like this Quix flaw is disclosed, you see exactly which sites are exposed in seconds, without opening a single admin panel to check.

Want the alerts and the tooling to act on them? Start with a free audit on one site and see your full extension inventory, or sign up for mySites.guru to get vulnerability alerts and one-click updates across every site you manage.

What to do right now

- Update every Joomla site running Quix Page Builder to **6.2.1** or later, one at a time or in bulk from one dashboard
- If you manage multiple sites, let mySites.guru show you which ones are still exposed rather than checking by hand
- Take a backup before updating, and clear caches afterwards
- If you were on a vulnerable version for a long time and handle sensitive data, treat any exposed data as potentially read, and if you suspect a breach, find any hacked files and backdoors and follow our guide to fixing a hacked Joomla or WordPress site

Disclosure and Severity

This flaw is CWE-89, SQL injection, reached by an anonymous visitor over the network in a single request, with no privileges and no user interaction. It is a read-only injection, so it stops short of the maximum score: an attacker reads the database but cannot

write to it or run code through this flaw. Reading the database is bad enough, because what leaks is every user record, every password hash, and the Joomla secret.

8.7
CVSS 4.0

HIGH **Our own assessment, pending the official vector**

Unauthenticated, exploitable over the internet with no user interaction, ending in full read access to the site database. It falls short of 10.0 only because it is a read, not a write: high confidentiality impact, but no integrity or availability impact. Being error-based makes exploitation trivial: the data returns in the response, one value per request.

- No login needed
- Exploitable over the internet
- No user interaction
- Full database read
- Password hashes exposed

A CVE for this vulnerability has been **requested through the Joomla CNA**, the body that assigns identifiers for Joomla and its extensions, crediting Phil Taylor of mySites.guru as the reporter. The identifier is pending and this post will be updated with it once assigned.

Field	Detail
CVE	Applied for via the Joomla CNA (pending)
Component	Quix Page Builder for Joomla (`com_quix`)
Vendor	ThemeXpert (themexpert.com)
Type	Unauthenticated, error-based SQL injection (CWE-89)
CVSS 4.0	8.7 (High), `AV:N/AC:L/AT:N/PR:N/UI:N/VC:H/VI:N/VA:N/SC:N/SI:N/SA:N` (our own assessment)
CWE	CWE-89 (Improper Neutralization of Special Elements used in an SQL Command)
Impact	Anonymous read access to any database table, including user accounts and password hashes
Finder	Phil Taylor, mySites.guru

Field	Detail
Affected versions	6.2.0 and all earlier versions
Fixed in	Quix 6.2.1

The disclosure ran on an unusually short cycle from audit to fix:

Date	Event
14 July 2026	ThemeXpert ships Quix 6.2.0, a release whose changelog opens “Completed a full security audit of the extension” and describes hardening every front-end API endpoint, input handling and output escaping, and file, URL and database query handling across the component.
15 July 2026	During routine security research on the extensions our customers rely on, mySites.guru audits that build and identifies the unauthenticated SQL injection, confirms it against the code, and proves it on a local test install that reads database contents back through a single anonymous request. The issue is disclosed privately to ThemeXpert alongside five further security findings, with all public detail withheld.
15 July 2026	ThemeXpert ships Quix 6.2.1 the same day, casting the article id to an integer and closing the injection, along with fixes for the other five issues we reported and hardening of the bundled JMedia media manager. mySites.guru adds Quix to its Joomla extension vulnerability database so every connected site below the fix is flagged, and requests a CVE through the Joomla CNA. No proof of concept is released.

Further Reading

- This Quix flaw was one of a month of Joomla extension vulnerabilities we found and disclosed - the full roundup of the run.
- Our DPCalendar SQL injection disclosure and AcyMailing SQL injection disclosure - the same class of bug, in different extensions, found in the same period.
- Why AJAX endpoints are a CMS security blind spot - the recurring public-endpoint pattern behind flaws like this one.

- How to check your Joomla database security - what to review and rotate if a vulnerable extension was ever live.
- CWE-89: SQL Injection and OWASP: SQL Injection - the canonical references for this weakness class.
- ThemeXpert - the vendor's site, where Quix 6.2.1 is available.

Frequently Asked Questions

What is the Quix Page Builder SQL injection vulnerability?

It is an unauthenticated SQL injection in Quix Page Builder, the drag-and-drop page builder for Joomla by ThemeXpert. One of Quix's front-end element endpoints took an article id from an anonymous request and placed it into a database query without turning it into a number first, letting an anonymous visitor inject conditions and read data from any table in the site's database. Because the endpoint reflects database errors back in its response, an attacker read the data straight out of the error message, one value per request. mySites.guru discovered the issue and reported it to ThemeXpert. It is fixed in Quix 6.2.1.

Do I need to log in to Joomla for an attacker to exploit this?

No. The affected element endpoint is reachable by anonymous visitors, with no account, no stolen credentials, and no CSRF token. That is what makes it serious. The main real-world limit is whether the site sits behind a web application firewall that filters SQL. Any site with Quix installed exposes the endpoint, so there is no second precondition to satisfy.

Which Quix versions are affected?

Quix 6.2.0 and every earlier version are affected. The vulnerable pre-check helper is long-standing and predates the hardening pass that fixed its sibling elements, so the whole Quix history below the fix is in range. ThemeXpert fixed it in Quix 6.2.1 by casting the id to an integer before it reaches the query. If you run any earlier version, update now.

How do I know if my site is affected?

If you run Quix Page Builder on a version earlier than 6.2.1, assume you are affected and update. mySites.guru flags every connected Joomla site running a vulnerable version automatically, so you get told which sites need attention rather than checking each one by hand.

Was this exploited in the wild before the fix?

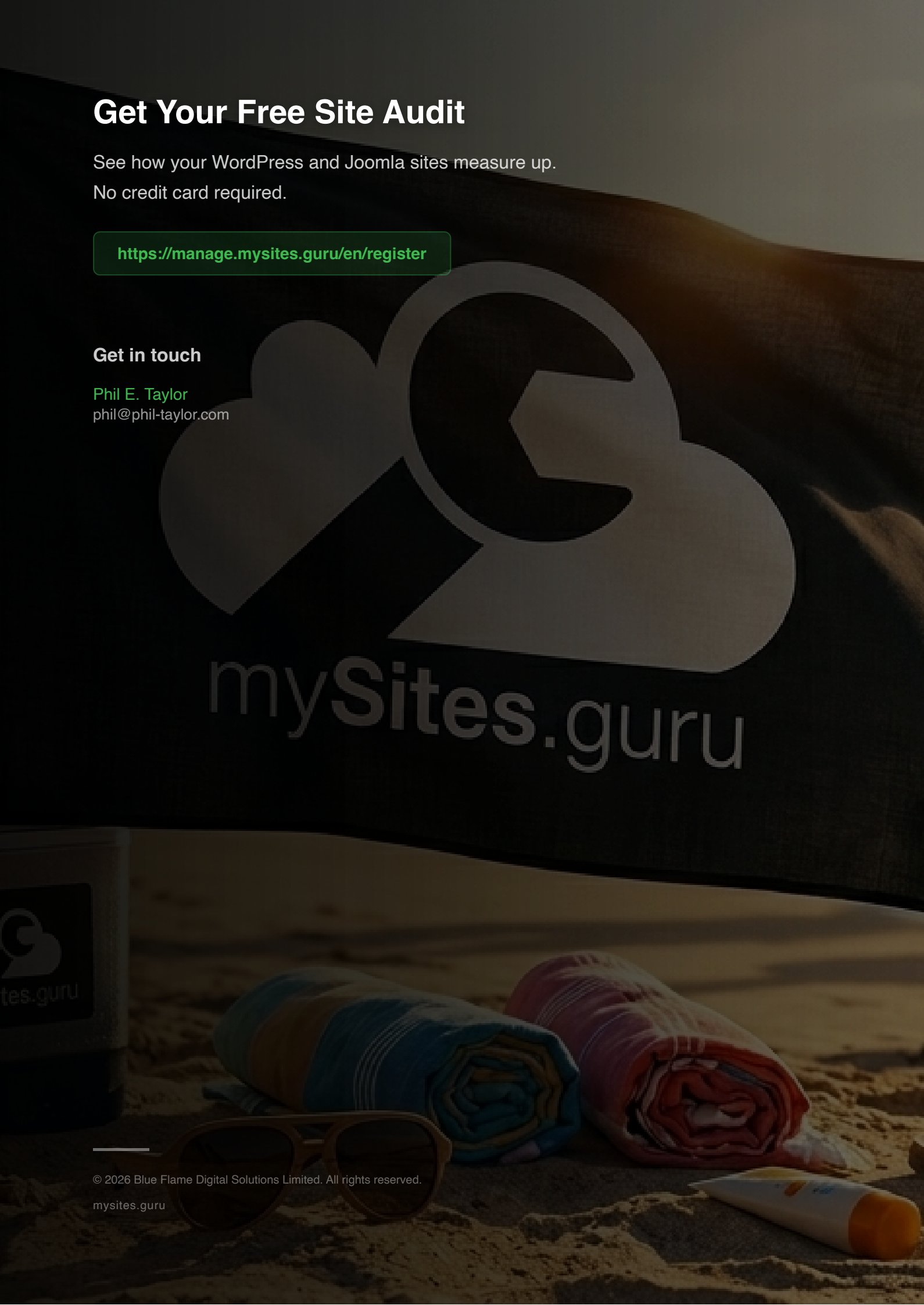
We have no evidence of exploitation before disclosure. Because the payload is SQL, a web application firewall that filters SQL blocks it. We reported the issue to ThemeXpert and withheld the exact request and any proof-of-concept, so the details were not public before the fix shipped.

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