mySites.guru

Cotton Cloud Patched the Login, Then the Data

Two access control flaws in Cotton Cloud for Joomla. The first fix closed the door and left the room unlocked. CVE-2026-67283 and CVE-2026-67284 are fixed in 2.0.3.

Phil E. Taylor | 12 August 2026

Cotton Cloud is a database-backed file storage system for Joomla. It ships as a package of six extensions and gives each user a private cloud drive inside the site: upload files, organise them into folders, edit text files in the browser, share them by link, and drive the whole thing from an AI-assisted terminal with about forty commands behind it.

In August 2026 **mySites.guru found that every front-end endpoint in Cotton Cloud 2.0.1 was reachable by a visitor who had never logged in**, and that from there they could read, overwrite, delete and re-permission every file any user had stored. We reported it privately. The developer shipped 2.0.2, then 2.0.3.

It took two releases, because the first one fixed one half of the report.

This post is about the half that was missed, about why that half is the half that always gets missed, and about how a fix can look completely correct in the diff and never execute once. That last part is where AI-assisted development comes in, and it is the bit worth your time even if you have never heard of this extension. Cotton Cloud 2.0.3 closes both halves, and if you run it that is the version you want.

TL;DR

- Cotton Cloud 2.0.1 had **two separate access control failures**, reported together on 1 August 2026
- **Missing authentication.** Every front-end task was gated by a CSRF token alone. Joomla hands those to guests, so an anonymous visitor could call all of them. This is CVE-2026-67283, CVSS 4.0 6.9 medium, fixed in 2.0.2
- **Missing authorisation.** The by-id database accessors fetched files and folders by numeric id with no check that the id belonged to the caller. This is CVE-2026-67284, CVSS 4.0 5.3 medium, fixed in 2.0.3

- **2.0.2 fixed the first and not the second.** The flaw stopped being anonymous and became exploitable by any registered user, which on a site with open signup is not much of a barrier
- 2.0.2 *did* contain an ownership check for the second half. It was added to a method **nothing in the package calls**, so it never executed. It read as a finished fix in the diff
- **2.0.3 closed the authorisation half properly**, with ownership checks through the model layer and a new assertion layer in front of the terminal and MCP surface
- **Update to 2.0.3.** Treat 2.0.2 and everything before it as vulnerable
- Both CVEs credit the finder, and the fix rounds were handled the way you would want a vendor to handle them

> mySites.guru discovered this issue and reported it privately before publishing anything. We withheld the exact requests and any proof-of-concept until a fix was available and site owners had a reasonable window to update. That included holding this post back through an incomplete fix, because publishing "this is fixed" while it is not is worse than publishing nothing.

What Cotton Cloud got right

Start here, because it matters and because parts of what follows are critical.

The developer engaged. After a slow start he replied, took the report seriously, shipped releases, and asked for further review each time. Plenty of vendors never reply at all. We have written about a vendor whose changelog said “Security Update” and nothing else, and about nineteen extension vulnerabilities we disclosed in a single month, so a developer who answers, ships, and then asks you to check his work is well ahead of the field.

The 2.0.2 fix for the authentication half is done the right way. Every task in both components gained a proper identity check:

```
$currentuser = Factory::getApplication()->getIdentity();

if (!$currentuser->get("id") || !Session::checkToken()) {
    echo new JsonResponse(null, Text::_('JINVALID_TOKEN'), true);
    return;
}
```

That is correct. `getIdentity()` returns a user object with id 0 for a guest, PHP short-circuits the `||`, and the request is refused before any logic runs. It was applied consistently across all twenty-one tasks in the file manager plus both entry points of the terminal component. Nothing was half-applied and nothing was missed. We verified it: an anonymous request that worked against 2.0.1 is refused by 2.0.2, and still is in 2.0.3.

2.0.2 also fixed something we had raised as a side issue. The 2.0.1 package shipped as 2.0.1 while every constituent manifest inside it declared `<version>2.0.0</version>`, which breaks version reporting for anyone trying to work out what they are running. Every manifest in 2.0.3 now carries the right number.

So: a developer who replied, fixed the authentication half cleanly, and came back and fixed the authorisation half. The problem is the gap in between, and that gap is the interesting part.

The half that was left in 2.0.2

Our report described two root causes, in two separate paragraphs, with the second one spelled out by name:

> *"The second is that the data layer only checks ownership in some places. The browse queries filter on owner_id, so the UI looks correctly scoped, but the by-id accessors do not.* `FileRepository::getForOpen()`, `::update()` *and* `::delete()` *all key on the id alone, as does* `CottonModel::file_update()`.
> `FolderManager::deleteRecursive()` *has no ownership check either."*

When 2.0.2 arrived we diffed it against 2.0.1. `FileRepository.php`, `FolderRepository.php`, `FolderManager.php` and `CottonBridge.php` were **byte-identical**. Every file named in that paragraph was untouched. The terminal's command interpreter had changed, but only by commenting out five debug logging calls.

Authentication was added. Authorisation was not.

The practical difference is smaller than it sounds. Cotton Cloud is a file store whose entire purpose is giving site users a private drive, so it is deployed on sites where people can register. Registration is free and instant. The attack went from "anyone on the internet" to "anyone on the internet who clicks Register first".

We confirmed all of the following against the shipping 2.0.2 on a local Joomla 5 lab, logged in as a plain self-registered user with no special permissions, targeting a file owned by the site administrator marked private with no sharing:

- **Read it.** The terminal's `cat` returned the full database row, file contents and all, along with the owner id proving it belonged to somebody else
- **Read it a second way.** The same content came back through the terminal's MCP interface, which exposed its tools to any logged-in user
- **Find it.** The `find` command enumerated another user's files and reported their folder paths
- **Overwrite it.** The save endpoint replaced the administrator's file contents
- **Delete it.** The delete endpoint removed an administrator-owned file permanently, not to the trash
- **Re-permission it.** The update endpoint set the file's sharing flag to public
- **Then read it with no account at all.** After the previous step, the file came back to a request carrying no cookies whatsoever

That last pair is the one to sit with. Cotton Cloud has exactly one endpoint with a correct, well-written access check, and it works. But in 2.0.2 a neighbouring endpoint with no check could flip the flag that the good endpoint reads. The correct ACL

was unlocked by the broken one, and the file ended up publicly readable by the entire internet.

The fix that never ran

Here is the part that changed how we read diffs.

Cotton Cloud 2.0.2 **did** add an ownership check for file deletion. It is right there in the diff, and it looks exactly like what you would want:

```
public function file_delete($file_id, $folder_id, $trash) {

    $currentuser = Factory::getApplication()->getIdentity();
    $user_id = (int) $currentuser->get("id");

    $del = $this->file_select($file_id);

    if (!$del->n || (int) $del->file[0]->owner_id !== $user_id) {
        $data->success = false;
        $data->error = Text::_('COM_COTTON_ERROR_NOACCESS');
        return $data;
    }
    // ...
```

Nothing calls that method. The controller task named `file_delete` does not call `CottonModel::file_delete()`. It calls `item_delete()`, a different method, which deleted by id with no ownership check anywhere in it. The method that got the fix was dead code, with no callers in the entire package. Every delete request the application actually served went down the unguarded path and always had.

It got worse on inspection. The check that was added did not even work: `file_select()` already filters by owner, so it can only return rows the caller owns, and the guard then asked whether each row is **not** owned by the caller, a condition that can never be true. If anything had called that method, it would have refused to delete your own files while reporting success.

An ownership check, written in good faith, sitting in code that cannot execute. Reading the diff would never catch it, because the diff looks right. Reading the diff tells you what you wrote. Only the request tells you what runs.

> Reproduce the bug before you fix it, then run the exact same request afterwards. A fix you have not exploited is a fix you have not tested. Reading the diff tells you what you wrote. Only the request tells you what runs.

One deletion request run as a second user against 2.0.2, watched succeeding, would have surfaced this in under a minute.

What vibe coding does to a security fix

A word of care first: we are not making a claim about how any particular line of Cotton Cloud was written, and none of this is aimed at its developer, who has been better to deal with than most. But the shape of what went wrong is worth naming, because we are running into it more and more, and this extension is a useful lens. Its headline feature is an AI terminal with an MCP server and around forty tool calls wired straight into the site's data layer.

A guard that is correct in isolation and unreachable in context is the signature failure of building fast with an AI assistant. The risk is structural, not one careless afternoon.

Generated code is written to the description, not to the request. Ask any assistant to "add an ownership check to file deletion" and it will find the method whose name says file deletion and put a very good check in it. It has no way of knowing the controller calls a differently named sibling, unless you make it trace the path first. The output is aimed at the wrong target, and aimed with complete confidence.

It reads as finished, which is what defeats review. Human mistakes usually look like mistakes. Generated mistakes look like the thing you asked for, because producing plausible code is precisely what the tool is optimised to do. The 2.0.2 guard is well-

formed, correctly indented, and uses the right Joomla APIs and the right language constant. Everything about it says “done” except whether it runs.

Structure arrives faster than the invariants that keep it safe. Cotton Cloud has repositories, managers, services and a bridge. That is a sensible architecture and a lot of scaffolding for one person to produce. Layers are exactly where security invariants go missing, because each layer assumes the one above it checked. Which is what happened: the browse queries filtered by owner, so the interface looked correctly scoped, while the by-id accessors underneath took an integer and returned whatever it pointed at.

The same helper gets written twice, and one copy gets fixed. The 2.0.3 changelog records removing a duplicate `SpaceCalculator` class that existed in both the library and the service layer. Duplication is cheap to generate and expensive to secure, because patching one copy leaves the other exactly as it was.

Plausible variable flow hides dead logic. The terminal’s JWT middleware tries three sources for its signing secret. The third reads the value into `$config` while the function returns `$secret`, so the fallback does nothing. It fails closed, which makes it latent rather than live, but nobody wrote that on purpose and nothing in the code admits to being broken.

An agent interface is new attack surface with no conventions yet. MCP has no authentication model of its own; it inherits whatever the host application enforces. Cotton Cloud’s MCP server handed its full tool list to anyone who could present a CSRF token, and every tool inherited the weakest check in the data layer beneath it. Each tool is an endpoint. Forty tools is forty endpoints, and they arrive as a single feature you tick on in the component options.

Use the assistants. What has to move is the acceptance test. When code was expensive to produce, reading it carefully was a fair proxy for knowing what it did, and most of our habits as reviewers were built on that assumption. When it is cheap to produce and uniformly plausible, the proxy stops working, and the only thing left that tells you the truth is running the request and watching what comes back. That cuts

both ways, and it is why we re-run the whole attack against a new release rather than reading the changelog and taking the vendor's word for it.

What 2.0.3 fixed

2.0.3 is a substantially bigger piece of work than 2.0.2 and it goes after the right thing.

Ownership checks now run through the model layer, on the paths a request actually takes: folder creation and updates, deletes, trash recovery, file uploads, finalisation, cancellation, saves, updates and the editor entry point all resolve the record and compare `owner_id` against the current user before doing anything with it.

The terminal and MCP surface, which was the widest part of the exposure because `CottonBridge` was built entirely on the unscoped accessors, gained a dedicated assertion layer:

```
public function assertFileOwner(int $fileId): ?array
{
    $file = $this->getById($fileId);
    if (!$file || (int) $file->owner_id !== $this->getCurrentUserId()) {
        return [
            'success' => false,
            'message' => Text::_('COM_SHUTTLE_ERROR_NOACCESS'),
            'error'   => Text::_('COM_SHUTTLE_ERROR_NOACCESS'),
        ];
    }

    return null;
}
```

That is the right shape: one place to get it right, called by the commands rather than reimplemented in each. Against 2.0.3, the reads we demonstrated through `cat`, `find` and the MCP interface are refused, and so are the file update and editor paths that gave us the re-permission and read primitives.

The changelog improved too, and that is worth its own paragraph further down.

We have seen this exact fix before

Here is the part that stops this being a story about one developer.

In June 2026 we reported a file-upload remote code execution flaw in PageBuilder CK. The vendor shipped 3.6.0, which **added a login check and nothing else**. Any user with an Editor account could still run code on the server, and that stayed true through 3.6.2 before it was finally fixed correctly in 3.6.3. Same shape: authentication added, authorisation forgotten, release announced as fixed.

Balbooa Forms did something adjacent. The first unauthenticated upload flaw was fixed in 2.4.1, and then a second one survived both 2.4.1 and 2.4.2 before 2.4.3 closed it.

Several extensions, several vendors, one reflex. When a report says “anyone can do X”, the instinct is to stop anyone from getting in. That closes the door the researcher walked through and leaves the room unlocked. It is an easy reflex to have because it makes the reproduction steps in the report stop working, which feels exactly like success.

The tell is always the same: if the fix is a check at the entrance and the report described a problem with the data, the fix is not finished. Missing authentication and missing authorisation are the same OWASP category and they need separate fixes. We see this often enough now that we test for it specifically: after a vendor tells us something is fixed, the first thing we do is log in as the lowest-privilege account the site will give us and run the whole attack again.

The IDOR half is not exotic either. We reported the same class of flaw in EasyStore, where any logged-in customer could read every other customer’s invoice by editing one URL, and in Events Booking, where registrant invoices with names, addresses and payment details were downloadable. Numeric ids, no ownership predicate, and an interface that looks correctly scoped because the listing query is.

Why half a fix happens

It is tempting to file this under carelessness. We do not think that is what it was, and the timeline says something more useful.

Date	What happened
1 August 2026	Reported privately to the developer, copying the Joomla Security Strike Team. Both root causes described, with the specific files named
8 August 2026	Resent after seven days of silence, having also tried to make contact publicly and privately on social media
10 August 2026	The Joomla Manual gained a Security page setting out twenty rules for handling exactly this situation
11 August 2026	First reply from the developer, ten days after the report. 2.0.2 had already been published, fixing the authentication half only
11 August 2026	The JSST asked us to confirm the fix. We confirmed it fixed one half
12 August 2026	The developer shipped 2.0.3 with the authorisation half addressed, and asked us to confirm it
12 August 2026	The Joomla CNA published CVE-2026-67283 for the authentication half and CVE-2026-67284 for the authorisation half

The first release is the one to study, because that is where the misunderstanding entered. There was no conversation between the report and 2.0.2. Ten days of silence, then a finished release and a reply announcing it. The developer's own summary of what he had done was that "improving security regarding task execution within the controllers was indeed necessary", which is an accurate description of the half he fixed and a complete description of what he had understood the report to be.

That is not carelessness. That is what happens when nobody talks. A single reply saying "I have read this, I understand there are two issues, here is my plan for each" would have surfaced the misunderstanding on day one, while the fix was still being

written. Instead the misunderstanding got shipped, and the person best placed to catch it found out afterwards, from a third party.

The second round went the way it should have. He engaged, shipped a real fix in a day, and asked us to verify before finalising. That is the loop working. It just started one release too late, and the reason it started late is worth more attention than the bug itself.

Where this sits against the twenty rules

The timing here is almost too neat. On 10 August 2026, the day before 2.0.2 was announced to us, David Jardin of the Joomla Security Strike Team added a Security page to the official Joomla Manual setting out twenty rules for how extension developers should handle a vulnerability report. We republished it in full, because it is the first time this guidance has existed in one citable place with the Joomla project's name on it.

The first release ran against several of them. We are listing them because the list is genuinely useful to any developer who has never had a report arrive in their inbox, not to keep score.

Rule 3, acknowledge the reporter quickly. The Manual's target is one business day for acknowledgement and three to five for an initial assessment. This took ten days and a resend.

Rule 4, validate the vulnerability. Half the report was never validated. This is the causal link and it is worth being blunt about: rules 3 and 4 are not manners, they are the steps where you find out what you are actually fixing. Skip them and you ship what you assumed the report said.

Rule 9, coordinate a disclosure date. No date was ever discussed. The release simply appeared, and we learned of it from the JSST thread rather than from the vendor.

Rule 11, assign the CVE before public disclosure. 2.0.2 went public first, and the identifiers followed afterwards.

Rule 17, do not hide security fixes in vague language. At the time 2.0.2 was serving to every site through the update system, the extension's changelog file contained exactly one entry, for version 1.0.1, reading:

```
Saving text files solved.
```

There was no entry for 2.0.2. Every site owner who clicked Changelog in Joomla to decide whether the update was urgent saw a note about text file saving from an ancient release, so a security release looked like an optional one.

This is the rule the developer has since fixed properly, and it deserves saying as loudly as the criticism. The changelog file now carries structured entries for every 2.x release, and the 2.0.3 entry says what the release did in plain terms:

```
Ownership validation layer added to CottonBridge
(assertFolderOwner, assertFileOwner, getCurrentUserId).
File owner assertions added to Shuttle commands (cat, head, and others).
```

A site owner reading that in Joomla's update view can tell it touches access control. Going back and writing the history you skipped is more work than writing it at the time, and he did it.

If you are an extension developer reading this and only take one thing from it: read the twenty rules. It is the Joomla project's own guidance, written by the person who runs the CNA that will assign your CVE, and following it costs you an email and a changelog line.

What site owners should do

If you run Cotton Cloud, update to 2.0.3 now. Do not stop at 2.0.2.

Then consider what was reachable while a vulnerable version was live. These flaws exposed and altered file contents, so updating closes the hole but does not undo any

read, overwrite or delete that already happened. If the extension was serving a site with open registration for any length of time, treat anything stored in it as potentially known. Because the flaw also allowed writing and deleting, check that the files and folders you expect are still there and still say what they should.

Also check for content that should not be there. The component sets the response content type from the file extension, so a file with an HTML extension could have been planted or edited to serve script from your domain, and it would look like an ordinary entry in a user's drive. Anything with an `.html`, `.htm` or `.svg` extension is worth opening.

> This is not remote code execution. Cotton Cloud stores files as database blobs rather than on disk, so a stored PHP file is returned as bytes, not executed. Nothing here lets an attacker run code on your server. What it does let them do is read, alter and destroy everything your users have stored, and serve script from your domain.

Finding every affected site at once

One Cotton Cloud install is an afternoon. Forty client sites is a different problem, and the version number is the only thing that tells you which ones matter.

mySites.guru records the installed version of every extension on every connected Joomla site, so a question like "which of my sites are running Cotton Cloud below 2.0.3" is a list you already have rather than an audit you have to run. When a version is flagged as vulnerable, the affected sites surface on their own.

That is the honest scope of it. We do not patch this for you and we cannot tell you what was read while a vulnerable version was live. What we remove is the part where you log into forty administrator panels to find out which four need attention. If you want the detail on how the version tracking works, we wrote it up in our guide to the vulnerable extension list.

For extension developers

Five things worth taking from this, none of which are specific to Cotton Cloud.

A CSRF token is not a login check. This is the single most common real finding we make in Joomla extensions. `Session::checkToken()` proves the request came from your site. It says nothing about who sent it, and Joomla issues tokens to guests. If the only gate on a task is a token, that task is anonymous.

Scope by owner in the query, not in the caller. Cotton Cloud's browse queries filtered by owner correctly, which is why the interface always looked right. The by-id lookups did not, and every caller that forgot to add its own check inherited a hole. Put the ownership predicate in the repository method where it cannot be forgotten, and the twenty places that call it are fixed at once.

Patch the path the request takes. Find the method the controller actually calls before you write the guard. In a layered codebase the method with the obvious name is often not the one doing the work. A guard in the wrong method is worse than no guard, because it looks done.

Give agent interfaces their own authorisation model. An MCP server or tool API inside your extension is not one endpoint, it is one endpoint per tool, and it inherits whatever the layer beneath it enforces. Decide who is allowed to call each tool, in the tool, and assume the transport gives you nothing.

Then prove it. Reproduce, patch, re-run the same request. Reading your own diff is how a fix that never executes gets shipped as done.

Further reading

- Twenty Rules for Joomla Extension Developers Handling a Security Report, the Joomla Manual's security page republished in full
- Joomla Manual: Security, the official source

- Joomla Security Strike Team, where to report an extension vulnerability and request a CVE
- OWASP: Broken Access Control, the category both halves of this flaw belong to
- OWASP: Insecure Direct Object Reference Prevention
- Helix3 and the changelog that said nothing, the same rule 17 failure in a much more widely installed extension
- PageBuilder CK RCE fixed, again, correctly this time, the same login-check-only fix in a different extension
- Nineteen and counting: a month of Joomla extension disclosures
- AJAX endpoints are a big CMS security blind spot, on why this class of endpoint keeps producing the same bug

Frequently Asked Questions

What was the Cotton Cloud vulnerability?

Cotton Cloud is a database-backed file storage system for Joomla. It had two separate access control failures. The first was missing authentication: every front-end task in com_cotton and com_shuttle was gated by a CSRF token alone, and Joomla issues those to visitors who have never logged in, so an anonymous visitor could reach all of them. The second was missing authorisation: the by-id database accessors looked up files and folders by their numeric id with no check that the id belonged to the person asking. Version 2.0.2 fixed the first. Version 2.0.3 fixed the second. The Joomla CNA assigned CVE-2026-67283 to the missing authentication and CVE-2026-67284 to the missing authorisation.

Is Cotton Cloud 2.0.2 safe?

No. 2.0.2 closed the anonymous route and left the underlying authorisation flaw in place, so any registered user could still read, overwrite, delete and re-permission other users' files. On a site with user registration open, which is the normal setup for a file store, a free account was all it took. Update to 2.0.3.

Which Cotton Cloud versions are affected?

Cotton Cloud 2.0.2 and earlier. 2.0.1 and earlier are exploitable by an anonymous visitor with no account at all, which is CVE-2026-67283. 2.0.2 narrowed that to any authenticated user, which is CVE-2026-67284. 2.0.3 closes both. If you are running anything below 2.0.3, update.

How do I know if my sites are affected?

If you run Cotton Cloud on a version earlier than 2.0.3, assume you are affected and update. mySites.guru tracks the installed version of every extension on every connected Joomla site, so you get told which sites need attention instead of logging into each one to check.

What is an IDOR, and why is it as serious as a missing login check?

IDOR stands for insecure direct object reference. It means the application fetches a record by its identifier without checking that the identifier belongs to the person asking. Cotton Cloud stored files with sequential numeric ids, so file 4 was simply the fourth file uploaded to the site by anyone. Asking for file 4 returned file 4. It is as serious as a missing login check because the login check is the only thing that was ever standing in front of it. Remove

the anonymous route and you still have every registered user able to address every other user's data by counting upwards.

What does this have to do with AI-assisted development?

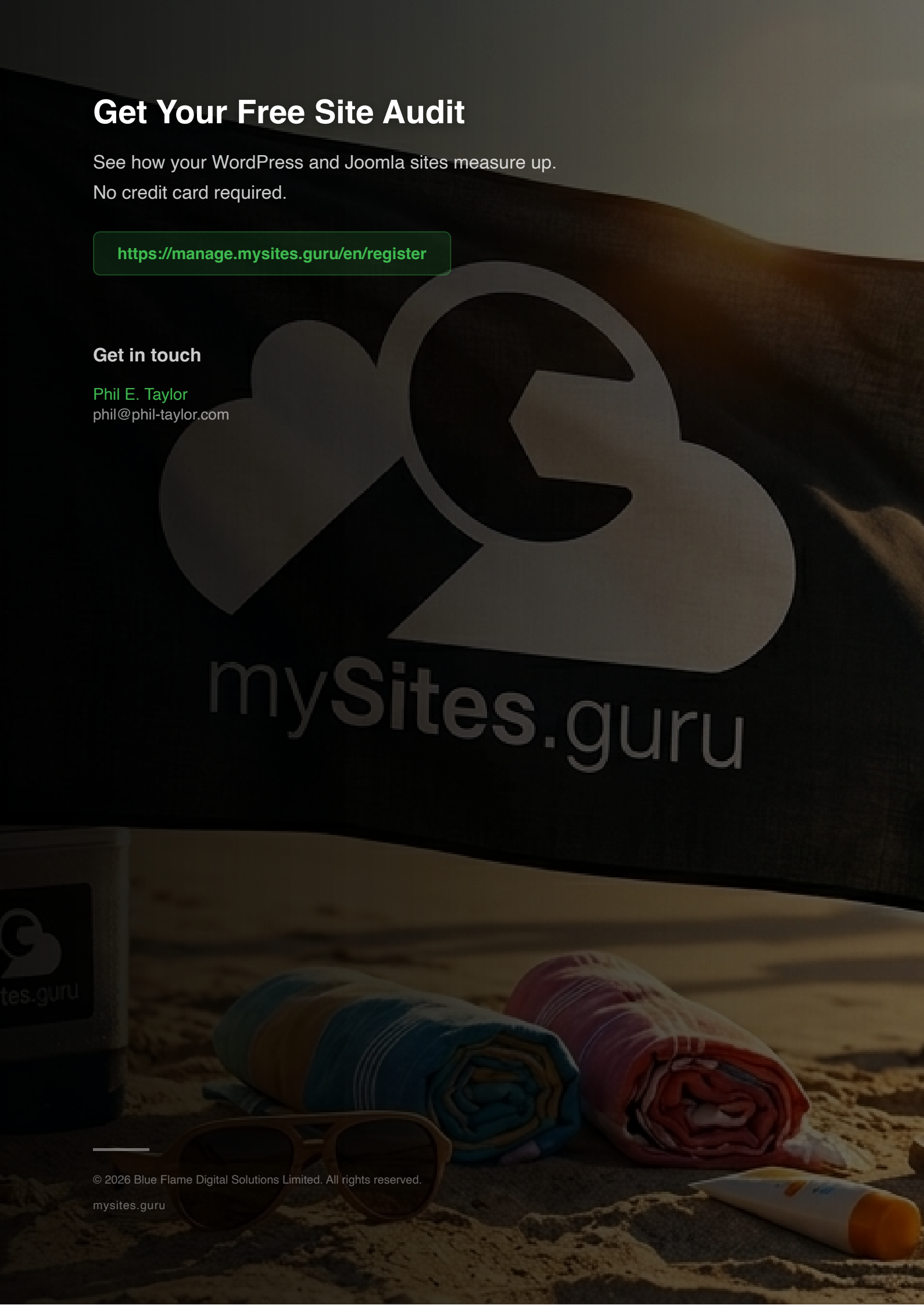
The first fix added an ownership check to a method that nothing in the codebase calls, so it never executed once. That specific failure, code that is correct in isolation and unreachable in context, is the characteristic risk of building fast with an AI assistant. A model writes to the description of the problem rather than to the path the request actually takes, and the result reads perfectly in a diff. The defence is to reproduce the bug, apply the fix, then re-run the identical request and watch it get refused, rather than trusting the code you have just read.

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