mySites.guru

Twenty Rules for Joomla Extension Developers Handling a Security Report

A new Joomla Manual page sets out 20 rules for how extension developers should handle a security report. Republished here in full under the JEDL.

Phil E. Taylor | 10 August 2026

The Joomla Manual now answers a question the Joomla ecosystem has never had a written answer to: when somebody emails you to say your extension has a security hole, what are you supposed to do next?

David Jardin, Team Leader of the Joomla Security Strike Team, wrote that answer as a single new file, `docs/building-extensions/security.md`. He opened it as pull request 695 on 10 August 2026 to help extension developers handle the security issues reported to them, and Harald Leithner merged it thirteen minutes later. It is documentation in the Manual now, not a proposal.

Twenty numbered practices. Publish a reporting channel. Keep the report confidential. Acknowledge the reporter inside a business day. Score it with CVSS 4.0 instead of guessing at "critical". Build the fix in private with a regression test. Get a CVE ID from the Joomla CNA before you go public. Credit the person who found it, in the words they chose. Publish an advisory that names affected and fixed versions. Do not hide the fix behind "various bug fixes and improvements". Do not sue the researcher.

None of that is exotic. It is the same coordinated-disclosure practice that the wider software industry settled on years ago, written down for the first time in a form Joomla extension developers can point at. We have reposted the whole thing below, in full, with credit to the author and the Joomla project, under the licence the Joomla Manual carries.

What this is, and what it is not

This is a **standard for handling security reports** written by **David Jardin**, Team Leader of the Joomla Security Strike Team, to help Joomla extension developers handle security issues reported to them. It arrived as pull request 695 on the Joomla Manual on 10 August 2026, was merged the same day, and is now published as Building Extensions: Security in the official Joomla developer documentation. It is not an enforced requirement: nothing checks whether an extension follows it. Our republication of it here is not endorsed by the Joomla

project. The full text below is the work of David Jardin and the Joomla! Project, republished with credit under the Joomla! Electronic Documentation License.

Why This Matters to Anyone Running Joomla Sites

We have spent this year on the reporting end of this process. In a single month we found and disclosed nineteen security issues in widely used Joomla extensions, and there have been many more since. That means we have watched the twenty points below get followed and skipped, one vendor at a time, in real cases with real site owners on the other end.

Two of those cases map onto the twenty points so exactly that they read like the worked examples.

Point 17 says do not hide security fixes in vague language. JoomShaper shipped Helix3 3.1.1 with a changelog entry that read, in its entirety, “Security Update”. That release closed an unauthenticated file write, an arbitrary file delete, a template-settings overwrite, an authenticated upload route to code execution, a stored XSS, a leaked API key and a ratings-abuse hole. Helix3 sits under a large share of the Joomla sites in the world. Every administrator looking at that line had to decide whether to schedule an update, and the line gave them nothing to decide with.

Points 11 and 12 say get the CVE before disclosure and always offer the reporter credit. We reported three flaws in JoomShaper’s EasyStore on 21 July 2026, including one where any logged-in customer could read every other customer’s invoice by editing a URL. The vendor acknowledged the next day, confirmed all three, and shipped the fix in 2.0.2 within 24 hours, which is the part of this they got right and deserve credit for. But the release carried no advisory, no coordinated disclosure date, no CVE request and no credit. We took the issues to the Joomla CNA ourselves, which assigned CVE-2026-65759, CVE-2026-65760 and CVE-2026-65761 on 23 July 2026.

In both cases the vulnerability itself was the least interesting part. Every codebase has them, which is exactly where the Manual page opens. What separated a decent

outcome from a bad one was the communication after the fix already existed, and that is the part a developer can settle permanently by writing a policy down once.

How Do You Know Which of Your Joomla Sites Are Affected?

A perfect advisory still only helps you if you can answer one question quickly: which of my sites run this extension, and on what version? For an agency with fifty Joomla sites that question currently means fifty administrator logins, and it has to be answered on the day the advisory is published, not the following week.

That is the job mySites.guru does. It keeps a live inventory of every extension on every connected Joomla and WordPress site, with the installed version on each, so a new advisory becomes one search rather than an afternoon. Our vulnerability database flags sites running a version a vendor has patched, including the extension-level issues that Joomla's own update system says nothing about, and the site audit covers the file and database checks that tell you whether an exposed site was reached before you got to it. If you are looking after more than a handful of sites, the multi-site Joomla tooling is the part of the subscription that turns an advisory into a work list.

The division of labour is worth being blunt about. Tooling tells you which sites to look at, but it cannot tell you a release was a security release when the vendor never said so. That half of the problem is the one only extension developers can solve, which is why the page below matters more than anything we could build.

What Does the New Manual Page Ask Developers to Do?

Read end to end, the twenty points fall into four groups.

Points 1 and 2 are the only two you can finish today, with no incident in progress. Publish a security contact and a policy that says how reports are received, how fast you respond, how disclosure is coordinated and whether you credit reporters. Put a

`SECURITY.md` in the repository, and say explicitly that reports should not go in the public issue tracker.

Once a report is live, points 3 to 8 take over. Acknowledge within one business day, assess within three to five, validate what is actually affected, score it with CVSS 4.0 rather than arbitrary labels, build the fix in a private branch with a regression test that fails before and passes after, and check whether the flaw reaches beyond your own code into a bundled library or another vendor's product.

Going public is points 9 to 15, and it is where Joomla extension vendors most often come unstuck. Agree a disclosure date rather than an indefinite embargo, request the CVE ID from `security@joomla.org` before you publish so the first advisory carries it, ask the reporter how they want to be credited and confirm the wording, publish a structured advisory with affected versions, fixed versions, severity, impact and timeline, and explain the flaw well enough for an administrator to assess their risk without handing out a working exploit.

The last five deal with the aftermath. Push the notice through channels people actually watch, keep the changelog honest, preserve an internal timeline, handle already-public reports without treating the researcher as hostile, and state in writing that good-faith research will not be met with legal threats.

The whole text follows.

The New Manual Page, Republished in Full

Source, author and licence

Everything from here to the end of point 20 is the work of **David Jardin**, Team Leader of the Joomla Security Strike Team, and the **Joomla! Project**, taken from `docs/building-extensions/security.md` as merged into the Joomla Manual by pull request 695 on 10 August 2026. It is republished here under the Joomla! Electronic Documentation License, which is the licence the Joomla Manual carries. No words have been changed. Documentation frontmatter and section rules were dropped, heading levels were mapped onto this page, four of the author's pull-quotes were restyled as callout boxes, and one

relative link was rewritten to its live address on manual.joomla.org. Full attribution, licence notice and source-form links are in the footer below.

Secure Coding Guidelines

For general information about security development of Joomla extensions, see the generic Security section.

Policies and Best Practices

It's barely a question *if* your extension will ever have a security vulnerability, it's just a matter of when. So, establishing security best practices, policies and workflows is a vital part of developing Joomla extensions.

1. Establish a clear security reporting channel

Every publicly available Joomla extension should provide an obvious way to report security vulnerabilities.

Recommended:

- A dedicated security contact such as `security@example.com`
- A `SECURITY.md` file in the source repository
- A security page on the vendor website
- Clear instructions on what information should be included
- An explicit statement that security reports should **not** be submitted through public issue trackers

The reporting channel should ideally support encrypted communication for sensitive reports.

A security policy should state:

- How reports are received
- Expected response times
- How vulnerabilities are handled
- How disclosure is coordinated
- Whether reporters are credited
- Whether CVE IDs are assigned

This follows the general principles of **ISO/IEC 29147**, which covers receiving vulnerability reports and communicating remediation information.

2. Treat security reports confidentially

A vulnerability report should initially be treated as confidential.

Do **not**:

- Create a public GitHub issue
- Publish the proof of concept
- Discuss the vulnerability publicly
- Include exploit details in public commit messages or changelogs

until a coordinated disclosure date has been agreed upon.

Internally, the report should receive a unique tracking ID and have a clearly defined owner.

A useful lifecycle is:

```
Reported
   ↓
Acknowledged
   ↓
Validated
   ↓
```

```
Severity assessed
   ↓
Fix developed
   ↓
Fix verified
   ↓
CVE assigned
   ↓
Coordinated disclosure
   ↓
Public advisory
   ↓
Post-disclosure monitoring
```

3. Acknowledge the reporter quickly

A reporter should receive an acknowledgement even if the vulnerability has not yet been validated.

Recommended targets:

- **Within 1 business day:** acknowledge receipt
- **Within 3–5 business days:** provide an initial assessment
- **During longer investigations:** provide regular status updates

The acknowledgement should explain:

- That the report was received
- Who is handling it
- Whether additional information is required
- What the next steps are

4. Validate the vulnerability

The maintainer should establish:

- Affected extension/component
- Affected versions
- Supported Joomla versions
- Attack prerequisites
- Required user privileges
- Whether authentication is required
- Whether user interaction is required
- Actual security impact
- Exploitability
- Whether the issue has already been publicly disclosed
- Whether another vendor or dependency is affected

The reporter should be given the opportunity to provide additional technical information or a proof of concept.

5. Distinguish security bugs from ordinary bugs

Not every bug is a security vulnerability.

Examples of typical security vulnerabilities include:

- SQL Injection
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Authentication bypass
- Authorization/access-control bypass
- Privilege escalation
- Arbitrary file upload
- Arbitrary file deletion
- Path traversal

- Remote or local code execution
- Sensitive information disclosure
- Insecure deserialization
- Server-Side Request Forgery (SSRF)

A vulnerability should be assessed based on its **security impact**, not merely on how severe the underlying coding mistake appears.

6. Assess severity consistently

Use a standardized scoring system rather than arbitrary labels such as “critical” or “minor”.

CVSS 4.0 is the recommended standard for communicating vulnerability severity.

At minimum, record:

```
CVSS:4.0/...
Base Score: 8.3
Severity: High
```

Also document the practical prerequisites, for example:

```
Attack Vector: Network
Privileges Required: Low
User Interaction: Required
Affected functionality: Administrator
Exploitability: Requires authenticated Joomla user
```

Do not treat the CVSS Base Score as the complete risk assessment. Consider the actual deployment environment and current threat situation as well.

7. Develop the fix privately

The fix should be developed without exposing the vulnerability.

Recommended practices:

- Private branch
- Private repository where necessary
- Security-specific test cases
- Regression tests
- Review by at least one additional developer
- Testing against all supported Joomla versions

The security fix should ideally include a regression test that:

1. Fails against the vulnerable implementation
2. Passes against the fixed implementation

8. Consider downstream dependencies

A Joomla extension may itself contain or depend on other software.

Before disclosure, check whether the vulnerability also affects:

- Joomla itself
- Another Joomla extension
- A third-party PHP library
- JavaScript dependencies
- An API or external service
- Another vendor's product
- Bundled or copied code

If multiple parties are affected, use **Coordinated Vulnerability Disclosure (CVD)** rather than disclosing independently.

9. Coordinate a disclosure date

The goal should not be to keep a vulnerability secret indefinitely.

Instead, agree on a reasonable disclosure timeline.

A practical model:

```
Day 0       Report received
Day 1       Acknowledgement
Day 3–5     Initial assessment
             ↓
             Fix development
             ↓
             CVE coordination
             ↓
             Fixed version released
             ↓
             Public advisory
```

The exact timeline should depend on severity and exploitability.

For a vulnerability that is already being exploited in the wild, the process should be accelerated substantially.

For vulnerabilities affecting several vendors, the disclosure date should be coordinated between all relevant parties.

10. CVE IDs

A **CVE ID identifies a vulnerability**, not a software release or a particular patch.

For example:

```
CVE-2026-12345
```

is associated with the vulnerability itself.

Do **not** create a separate CVE merely because different releases address the same vulnerability.

Separate vulnerabilities should normally receive separate CVE IDs.

Joomla extensions

Joomla extension developers can request a CVE ID from the Joomla security team by contacting:

```
security@joomla.org
```

The CVE assignment should be coordinated with Joomla's **CVE Numbering Authority (CNA)** rather than inventing or requesting an arbitrary CVE number.

11. Assign the CVE before public disclosure

Ideally, the first public security advisory should already contain the CVE ID.

Extension developers can request the CVE ID from:

```
security@joomla.org
```

before public disclosure so that it can be included in the initial announcement.

The CVE ID should be associated with the vulnerability and consistently referenced across:

- Security advisory
- Changelog
- Release notes
- CVE record

- Joomla security communication
- Other relevant security databases

Follow the current CVE Program and CNA publication requirements and coordinate embargoes where applicable.

12. Credit the security researcher

Always offer appropriate credit to the reporter.

For example:

```
This vulnerability was responsibly reported by
Jane Doe (@janedoe).

We thank Jane for responsibly reporting this issue
and helping us improve the security of this extension.
```

However:

> Never publish a reporter's name, handle, company or other identifying information without their consent.

Ask the reporter exactly how they want to be credited.

Possible choices include:

```
Name:
Jane Doe

Handle:
@janedoe

Organization:
Security Research Lab
```

```
Anonymous:
Yes
```

Confirm that the proposed credit is acceptable before publication.

13. Publish a security advisory

Do not rely exclusively on a generic changelog such as:

```
1.4.2
– Various bug fixes and improvements
```

Publish a dedicated security advisory.

A proper advisory should contain:

```
Security Advisory

Product
Example Joomla Extension

Vulnerability
Stored Cross-Site Scripting

CVE
CVE-2026-12345

Affected Versions
1.0.0 – 1.4.1

Fixed Versions
1.4.2 and later

Severity
High
```

```
CVSS
CVSS:4.0/...

Impact
An authenticated user with ... can ...

Description
...

Solution
Update to version 1.4.2 or later.

Credits
Reported by Jane Doe.

Timeline
2026-07-01  Report received
2026-07-02  Vulnerability confirmed
2026-07-15  Fix released
2026-07-15  Advisory published
```

Joomla's own security advisories provide a useful model: they identify affected versions, impact, severity, exploit type, report/fix dates, CVE number, description, solution and reporter.

14. Tell users what they need to do

The most important information for users is:

> Am I affected, and what should I do?

Every advisory should clearly state:

Affected versions

```
Versions 2.0.0 through 2.4.7 are affected.
```

Fixed versions

```
Upgrade to 2.4.8 or later.
```

Recommended action

```
All users running an affected version should update immediately.
```

If updating is not possible, provide a mitigation where one genuinely exists.

For example:

```
Until the extension can be updated, disable the affected
administrator endpoint.
```

Do not recommend a mitigation that has not been tested.

15. Explain the vulnerability without unnecessarily publishing an exploit

A security advisory should provide enough information for administrators to understand the risk.

It should answer:

- What is the vulnerability?
- Who can exploit it?
- Is authentication required?
- What can an attacker accomplish?
- Which versions are affected?
- Is exploitation known?
- How can users fix it?

A vendor does **not necessarily need to publish a working exploit/PoC** immediately.

Especially for high-impact vulnerabilities, avoid publishing exploit details that materially increase the risk to users who have not yet updated.

The principle should be:

> **Transparency sufficient for risk assessment, without unnecessarily increasing exploitability.**

16. Notify users through appropriate channels

Security fixes should be communicated through channels users actually monitor.

Depending on the extension:

- Extension update mechanism
- Joomla Extension Directory
- Vendor website
- Security advisory page
- Mailing list
- RSS feed
- GitHub Security Advisory
- Newsletter
- Social media, where appropriate

The Joomla Security Centre provides a public security announcement feed that users can subscribe to.

For particularly severe vulnerabilities, do not rely exclusively on a changelog.

17. Do not hide security fixes in vague language

Avoid:

```
Various bug fixes and improvements
```

when a security vulnerability was fixed.

Prefer:

```
Security: Fixed an authenticated SQL injection vulnerability
in the administrator filtering functionality.

CVE-2026-12345
```

Security transparency helps users make informed decisions about updates.

18. Preserve the disclosure timeline

Maintain an internal timeline for every security issue:

```
2026-07-01  Vulnerability reported
2026-07-01  Report acknowledged
2026-07-03  Vulnerability confirmed
2026-07-04  CVE requested
2026-07-08  Fix completed
2026-07-10  Fix provided to reporter for verification
2026-07-12  Fixed release published
2026-07-12  CVE published
2026-07-12  Security advisory published
```

This is useful for:

- Internal audits
- Communication with researchers
- CVE records

- Incident response
- Process improvements

19. Handle reports that are already public

Sometimes a researcher publishes a vulnerability before contacting the vendor.

Do not automatically treat this as malicious.

Instead:

1. Acknowledge the report
2. Assess the vulnerability
3. Determine whether a CVE already exists
4. Coordinate with the reporter where possible
5. Release a fix as quickly as practical
6. Publish an advisory

The CVE process also provides mechanisms for vulnerabilities that have already been publicly disclosed.

20. Do not retaliate against good-faith researchers

A responsible disclosure process should explicitly state that good-faith security research is welcome.

A useful policy statement is:

> We welcome good-faith security research and responsible vulnerability disclosure. Researchers who follow this policy will not be subject to legal action by us solely for activities conducted in accordance with this policy.

However, define reasonable boundaries around:

- Accessing other users' data
- Destructive testing
- Denial-of-service testing
- Social engineering
- Automated scanning of third-party infrastructure
- Persistence
- Data exfiltration

Credit, Licence and the Pull Request

> **End of the republished work**
>
> The twenty numbered practices above are the work of David Jardin and the Joomla! Project, not of mySites.guru.

Author. David Jardin, Team Leader of the Joomla Security Strike Team, the group that receives security reports for the project and operates Joomla's CVE Numbering Authority. He contributes to the Joomla repositories as SniperSister, and has led the JSST since 2021. This document was written to help Joomla extension developers handle security issues reported to them.

Source. joomla/Manual pull request 695, opened and merged on 10 August 2026, adding `docs/building-extensions/security.md`. The source form of the work, in the plain Markdown it was written in, is available free of charge over the internet: read the raw file on the Manual's `main` branch, read the exact revision republished here at merge commit `a091273`, or browse the diff. The merged text is identical to what is reproduced above. It is now published on the documentation site as Building Extensions: Security, currently under the `next` documentation version. This page is

itself available in Markdown source form by requesting it with an `Accept: text/markdown` header.

Licence. The Joomla Manual is published under the Joomla! Electronic Documentation License (full text in the repository). That licence applies to the republished work above. As its conditional permissions require, mySites.guru grants a Grantback License to the author and a Copyleft License to everyone who receives this page, so the Joomla project may take this material back into its authoritative documentation and anyone downstream may propagate it on the same terms. The JEDL covers electronic media only, so reproducing it in print needs separate permission from the Joomla project.

Status. Pull request 695 was merged into the Joomla Manual's `main` branch on 10 August 2026 at 11:05 UTC, thirteen minutes after David opened it, by Harald Leithner. The text above is therefore official Joomla developer documentation rather than a proposal. It is not an enforced requirement: nothing checks whether an extension follows it, and no Joomla Extensions Directory rule is attached to it. Separately, the JEDL grants no right to represent republished material as official or endorsed, so to be clear, the upstream Manual page is official Joomla documentation and this republication of it is not endorsed by the Joomla project.

If you maintain a Joomla extension, the merge is not the end of the conversation. Points 3 and 9, the response-time targets and the disclosure timeline, are the ones most likely to need a reality check from people shipping extensions with small teams, and a one-day acknowledgement target reads differently when the maintainer is one person with a day job. That feedback now belongs in a fresh issue or pull request on the joomla/Manual repository rather than a review comment on a closed pull request.

Our own view, from the reporting side: point 1 is the one to act on this week. A `SECURITY.md` and a published contact address costs an afternoon, works before you ever have an incident, and is the difference between a researcher reaching you privately and a researcher giving up and publishing.

Further Reading

- Joomla Manual: Building Extensions, Security - the page reproduced above, read at source on the official documentation site.
- Joomla Manual: Security fundamentals - the secure-coding half of the picture, covering the practices that stop the vulnerability existing in the first place.
- Joomla Developer Network: Security - the Joomla Security Strike Team's own pages, including the security announcement feed point 16 refers to.
- Joomla! Electronic Documentation License - the licence this page republishes under, and the one that lets anyone else do the same.
- FIRST: CVSS 4.0 specification - the scoring standard point 6 recommends, from the body that publishes it.
- ISO/IEC 29147:2018 - "Vulnerability disclosure", the international standard covering how a vendor receives vulnerability reports and publishes remediation information. Point 1 draws its principles from it, and the 2018 second edition is the current one.
- CVE Program: CNA rules - the publication requirements point 11 tells developers to follow.
- CVE Program partner list - where the Joomla! Project's CNA scope is recorded as "Core Joomla! CMS, the Joomla Framework, and Joomla! Extensions issues only".

If you look after Joomla sites rather than build the extensions on them, the practical companion to this is knowing which of your sites run what. Try a free audit on one site, or read our roundup of nineteen Joomla extension vulnerabilities disclosed in a month for what the reporting side of this process actually looks like.

Frequently Asked Questions

What is the new Joomla extension security page?

It is a Security page added to the Joomla Manual, the project's official developer documentation. It sets out 20 numbered practices for handling a vulnerability report in a Joomla extension, covering how to publish a reporting channel, treat a report confidentially, acknowledge the reporter, assess severity with CVSS 4.0, develop the fix privately, obtain a CVE ID through the Joomla CNA, credit the researcher, publish a real advisory and preserve the disclosure timeline. It arrived as pull request 695 on the joomla/Manual repository on 10 August 2026 and was merged the same day.

Is this binding on Joomla extension developers?

No. It is documented best practice in the official Joomla Manual, not an enforced requirement. Nothing checks whether an extension follows it, and there is no Joomla Extensions Directory rule attached to it. What changed is that the guidance now exists in one citable place with the Joomla project's name on it, so a developer can point at it and a site owner can measure a vendor against it.

Who wrote it?

David Jardin, who submits to the Joomla repositories as SniperSister, opened the pull request on 10 August 2026. He is Team Leader of the Joomla Security Strike Team, the group that handles security reports for the project and operates Joomla's CVE Numbering Authority, and has led that team since 2021. Harald Leithner merged it the same day. David wrote it to help Joomla extension developers handle security issues reported to them.

How does a Joomla extension developer get a CVE ID?

By emailing security@joomla.org. Joomla operates its own CVE Numbering Authority, and the CVE Program lists its scope as "Core Joomla! CMS, the Joomla Framework, and Joomla! Extensions issues only", so third-party extensions are covered and developers do not need to go elsewhere or invent an identifier. The Manual page tells developers to request the ID before public disclosure so the first advisory can carry it.

What does the standard say about crediting the person who reported the flaw?

It says credit should always be offered, and that a reporter's name, handle, company or other identifying details must never be published without their consent. The documented

practice is to ask the reporter how they want to be credited, offering name, handle, organisation or anonymous, and to confirm the wording before publication.

Why does a changelog line saying "various bug fixes and improvements" matter?

Because it is the difference between a site owner updating today and updating in three months. Point 17 of the new Manual page says not to hide security fixes in vague language. We have written up real cases of this: Helix3 3.1.1 shipped as a single changelog entry reading "Security Update" while closing an unauthenticated file write, an arbitrary file delete, a stored XSS and a leaked API key. Nobody reading that line could tell it was urgent.

Can I republish this content myself?

Yes, on electronic media. The Joomla Manual is licensed under the Joomla! Electronic Documentation License. It permits republishing for any purpose including commercial, provided you grant a Grantback License to the author and a Copyleft License to your own readers, make the source form available free of charge, include the licence and a notice that it applies, preserve the author's copyright notices as prominently as your own, and do not represent the work as official or endorsed. Print is not covered.

Does the standard tell site owners anything, or is it only for developers?

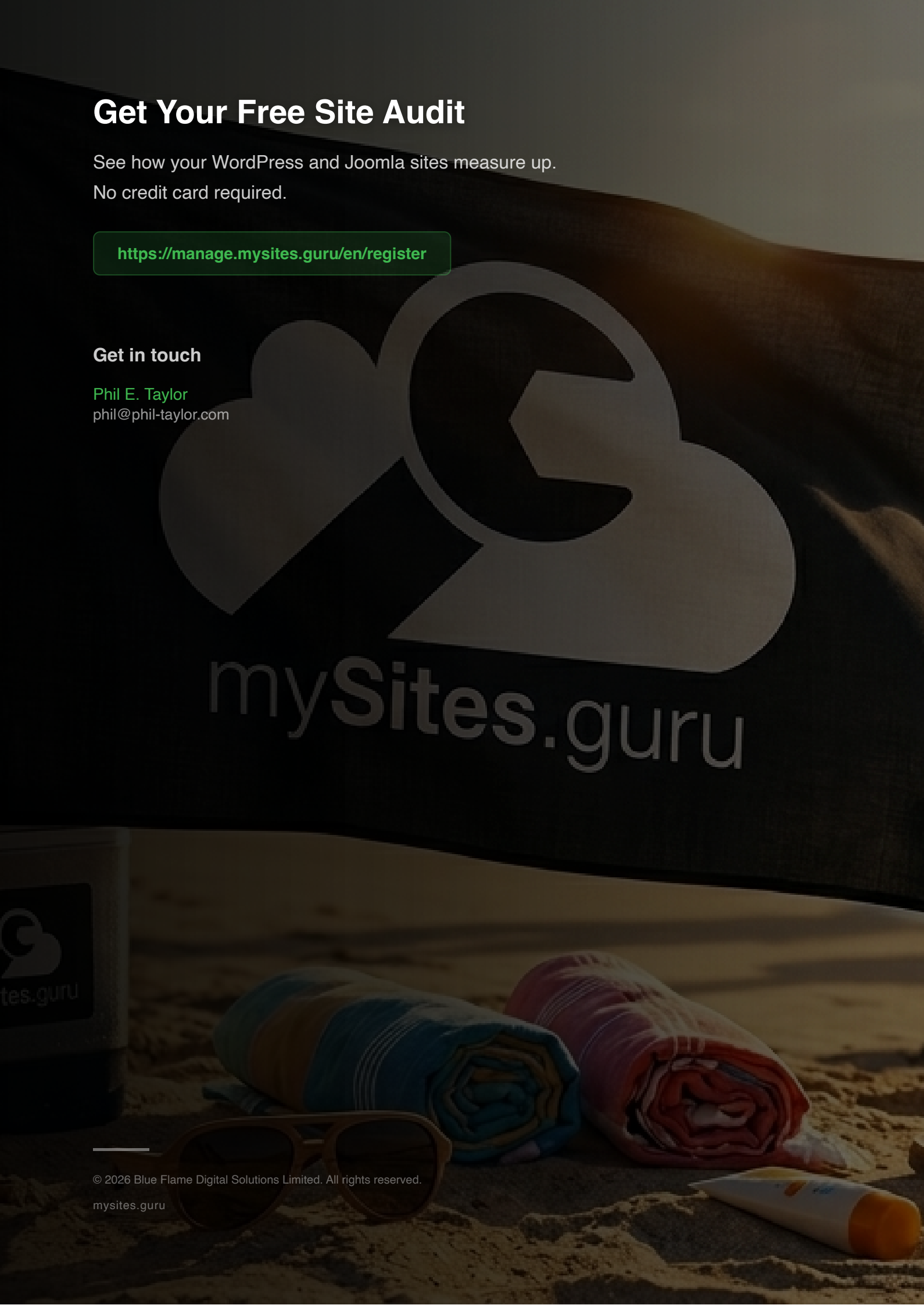
It is written for extension developers, but points 13 through 17 define what a site owner should expect to receive: a dedicated advisory naming affected and fixed versions, a plain statement of what to do, a CVE ID, and a changelog that does not disguise the fix. If a vendor you rely on publishes none of that, this page is now a citable yardstick to hold them to, with the Joomla project's name on it.

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