Blogs Posts Full Account Takeover for All-Roles & Privilege-Escalation via Stored DOM XSS

Critical **bertugfahriozer** published **GHSA-x7wh-g25g-53vg** last week

Package

php **ci4-cms-erp/ci4ms** (Composer)

Affected versions	Patched versions
<= 0.28.6.0	0.31.0.0

Description

Summary

Vulnerability: Stored DOM XSS via Blog Post Content (Persistent Payload Injection)

- Stored Cross-Site Scripting via Unsanitized Blog Post Content in Blog Management

Description

The application fails to properly sanitize user-controlled input when creating or editing blog posts. An attacker can inject a malicious JavaScript payload into blog post content, which is then stored server-side.

This stored payload is later rendered unsafely in multiple application views without proper output encoding, leading to stored cross-site scripting (XSS).

Affected Functionality

- Blog post creation functionality
- Blog post editing functionality
- Blog post storage and retrieval logic

Attack Scenario

- An attacker creates or edits a blog post to include a malicious XSS payload.
- The application stores this content without sanitization or encoding.
- The payload persists and executes whenever the blog post is rendered in affected views.

Impact

- Persistent Stored XSS
- Execution of arbitrary JavaScript in victims' browsers
- Privilege escalation when viewed by administrators or privileged users
- Full administrator account takeover
- Full account takeover across all roles
- Full compromise of the entire application

Endpoints:

- `/backend/blogs/create`
- `/backend/blogs/`
- `/blog/{id}`

Steps To Reproduce (POC)

1. Go to the Blog Post Create or Edit page
2. Insert an XSS payload into the blog post content such as:
   `<img src=x onerror=alert(document.domain)>`
3. Save or publish the blog post
4. View the post via the administrative panel or public blog page
5. Notice the XSS payload executing automatically

Remediation

- Never use .html() again or any innerHTML-style like JS in your PHP, or any other sink, even if user inputs that flow into them are not clear, they still represent real world danger as an attacker can make use of this to exploit the application via XSS. And do HTML Encoding as much as possible and always do Sanitization, theres no sanitization there unfortunately. Also apply CSP, HttpOnly, SameSite, and Secure upon all application, they reduce severity of XSS & escalated-CSRF via XSS and do great jobs

Ready Video POC:

https://mega.nz/file/bYtCQRqT#ph1S_01XaYXiNTzanP3AVL6aQMe0YC5Py7Gko1FoT4A

Severity

Critical 9.1 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	Low
Availability	Low

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:L/A:L

CVE ID

CVE-2026-34568

Weaknesses

▶ CWE-79

Credits

 **bugmithlegend** Reporter

 **peeefour** Reporter