Stored XSS via Unanchored Duration Regex in Video Encoder Receiver

Moderate **DanielnetoDotCom** published **GHSA-8pv3-29pp-pf8f** last week

Package

php **wwbn/avideo** (Composer)

Affected versions

<= 29.0

Patched versions

None

Description

Summary

The `isValidDuration()` regex at `objects/video.php:918` uses `/^[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}/` without a `$` end anchor, allowing arbitrary HTML/JavaScript to be appended after a valid duration prefix. The crafted duration is stored in the database and rendered without HTML escaping via `echo Video::getCleanDuration()` on trending pages, playlist pages, and video gallery thumbnails, resulting in stored cross-site scripting.

Details

Input entry point: `objects/aVideoEncoderReceiveImage.json.php:208`

```
// Line 203-211
if (!empty($_REQUEST['duration'])) {
    $video->setDuration($_REQUEST['duration']);
}
```

Insufficient validation: `objects/video.php:918`

```
static function isValidDuration($duration) {
    // ...
    return preg_match('/^[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}/', $duration);
```

```
    //     Missing $ anchor here ----------------------------------^
}
```

The regex matches `00:00:01` at the start of the string but ignores everything after it. A payload like `00:00:01</time><img src=x onerror=alert(1)><time>` passes validation.

No sanitization in output function: `objects/video.php:3463-3480`

```
public static function getCleanDuration($duration = "") {
    $durationParts = explode(".", $duration);
    $duration = $durationParts[0];
    $durationParts = explode(':', $duration);
    if (count($durationParts) == 1) {
        return '0:00:' . static::addZero($durationParts[0]);
    } elseif (count($durationParts) == 2) {
        return '0:' . static::addZero($durationParts[0]) . ':' . static::addZero($durati
    }
    return $duration; // Returns full string unmodified for 3+ colon parts
}
```

With the payload `00:00:01</time><img src=x onerror=alert(1)><time>`, exploding by `:` yields 3+ parts, so the full unsanitized string is returned.

Unescaped output sinks:

1. `view/trending.php:72`:

```
<time class="duration"><?php echo Video::getCleanDuration($value['duration']); ?><
```

2. `view/include/playlist.php:159`:

```
<time class="duration"><?php echo Video::getCleanDuration(@$value['duration']); ?>e
```

3. `objects/video.php:7200` (gallery thumbnail generation):

```
$img .= "<time class=\"duration\"...>" . $duration . "</time>";
```

No Content-Security-Policy headers are set. The application uses raw PHP templates with no auto-escaping framework.

PoC

1. Authenticate as a user with upload permission and obtain a `video_id_hash` for a video (visible in encoder API responses or via the upload flow).

2. Send the malicious duration:

```
curl -X POST "https://target/objects/aVideoEncoderReceiveImage.json.php" \
  -d "videos_id=VIDEO_ID" \
  -d "video_id_hash=HASH" \
  -d 'duration=00:00:01</time><img src=x onerror=alert(document.cookie)><time>'
```

3. The `isValidDuration()` regex matches the `00:00:01` prefix and allows the full string to be stored.

4. Visit the trending page (`/view/trending.php`) or any playlist containing the poisoned video. The injected HTML breaks out of the `<time>` tag and the `onerror` handler executes JavaScript in the victim's browser context.

Impact

- **Session hijacking**: Attacker can steal session cookies of any user (including administrators) who views a page listing the poisoned video (trending, playlists, search results, channel pages).
- **Account takeover**: Stolen admin session cookies grant full platform control.
- **Phishing**: Attacker can inject fake login forms or redirect users to malicious sites.
- **Worm potential**: Since the XSS fires on commonly-visited listing pages (trending), it can propagate without targeted delivery — any visitor is a victim.

The attack requires only upload-level permissions (low privilege) and impacts all users who view any page rendering the poisoned video's duration (high blast radius).

Recommended Fix

Fix 1 — Anchor the regex (`objects/video.php:918`):

```
- return preg_match('/^[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}/', $duration);
+ return preg_match('/^[0-9]{1,2}:[0-9]{1,2}:[0-9]{1,2}(\.[0-9]+)?$/', $duration);
```

Fix 2 — HTML-escape all duration output (defense in depth):

In `view/trending.php:72`:

```
- <time class="duration"><?php echo Video::getCleanDuration($value['duration']); ?>
+ <time class="duration"><?php echo htmlspecialchars(Video::getCleanDuration($value[
```

In `view/include/playlist.php:159`:

```
- <time class="duration"><?php echo Video::getCleanDuration(@$value['duration']); ?></time>
+ <time class="duration"><?php echo htmlspecialchars(Video::getCleanDuration(@$value['du
```

In `objects/video.php:7200`:

```
- $img .= "<time class=\"duration\"...>" . $duration . "</time>";
+ $img .= "<time class=\"duration\"...>" . htmlspecialchars($duration, ENT_QUOTES,
```

Both fixes should be applied: the regex fix prevents storage of invalid data, and the output escaping provides defense in depth against any other code path that might store unvalidated durations.

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-2026-41061

Weaknesses

▶ CWE-79

Credits

offset Reporter