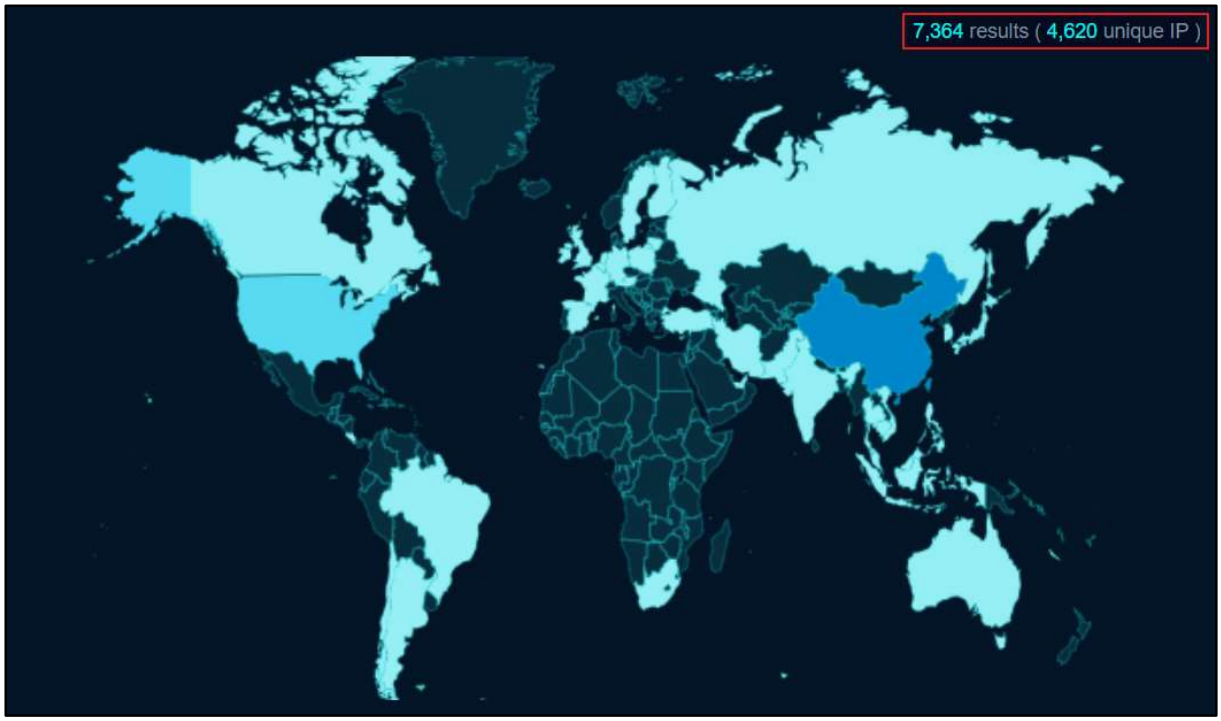
Research & Technique

Lobe Chat SSRF 취약점(CVE-2024-47066)

■ 취약점 개요

Lobe Chat 은 오픈 소스로 제공되는 최신 디자인 통합 LLM[1] 프론트엔드[2] 프레임워크다. 다양한 플러그인 기능을 지원하며, OpenAI 의 ChatGPT, Claude, Gemini, Groq, Ollama 등 다양한 AI 모델 및 플랫폼을 활용한 채팅 애플리케이션을 무료 배포할 수 있다.

OSINT[3] 검색 엔진을 통해 인터넷 상에 공개된 Lobe Chat 을 조회한 결과, 2024 년 10 월 1 일 기준 중국과 미국을 비롯한 수많은 국가의 7 천여 개 사이트에서 Lobe Chat 을 배포하는 것이 확인됐다.



출처: fofa.info

그림 1. Lobe Chat 사용 통계

2024 년 9 월 23 일, Lobe Chat 의 SSRF(Server-Side Request Forgery) 취약점(CVE-2024-47066)이 공개됐다. 해당 취약점은 애플리케이션 내 요청 기능에서 실제 요청하는 IP 주소에 대한 검증이 미흡할 시 발생된다.

---

[1] LLM(Large Language Models): 텍스트를 인식하고 생성하는 등의 작업을 수행할 수 있는 일종의 인공지능(AI) 프로그램

[2] 프론트엔드(Front-end): 웹사이트나 앱 등의 사용자 인터페이스(UI)를 개발하는 분야

[3] OSINT(Open Source INTelligence): 공개된 출처에서 합법적으로 수집한 정보

외부 서비스 주소 입력 시, 내부망 주소로 리다이렉션[4] 되는 경우 입력 주소 검증 과정을 우회할 수 있다. 2024 년 3 월 11 일, LLM 지원 크로스 플랫폼 채팅 애플리케이션 NextChat 에서도 유사한 SSRF 취약점(CVE-2023-49785)이 공개되었다. 이에 LLM 프론트엔드 애플리케이션 사용 시 SSRF 취약점이 발생하는지 확인할 필요가 있다.

이를 이용해 공격자는 SSRF 취약점을 통해 내부에서만 접근할 수 있는 민감 데이터에 접근할 수 있고, 일부 상황에서는 SSRF 취약점을 이용하여 임의의 명령 실행을 수행할 수 있다.

■ 공격 시나리오

CVE-2024-47066 의 공격 시나리오는 아래와 같다.

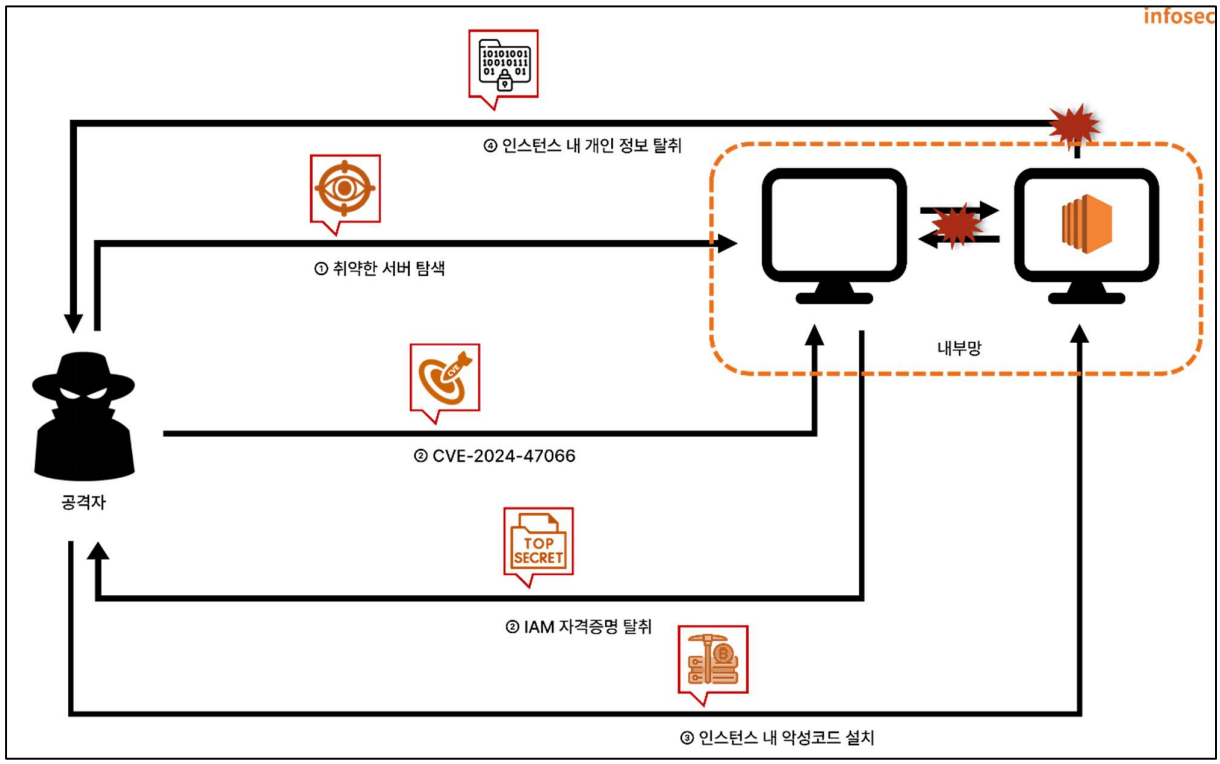


그림 2. CVE-2024-47066 공격 시나리오

① 공격자는 LLM 프론트엔드 프레임워크로 Lobe Chat을 사용 중인 취약한 서버 탐색
② 공격자는 CVE-2024-47066 취약점을 이용하여 EC2 인스턴스 IAM 자격증명 탈취
③ 공격자는 탈취한 EC2 인스턴스 IAM 자격증명으로 해당 EC2 인스턴스에 악성 코드 설치
④ 공격자는 서버에 설치된 악성 코드로 중요 정보 탈취

[4] 리다이렉션(Redirection): HTTP 에서 리다이렉션은 3xx 상태 코드를 지닌 응답으로, 수신한 브라우저는 제공된 새로운 URL 을 사용하여 즉시 로드한다.

■ 영향받는 소프트웨어 버전

CVE-2024-47066 에 취약한 소프트웨어 버전은 다음과 같다.

S/W 구분	취약 버전
Lobe Chat	1.19.12 이전

■ 테스트 환경 구성 정보

테스트 환경을 구축하여 CVE-2024-47066 의 동작 과정을 살펴본다.

이름	정보
피해자	Lobe Chat 1.19.12 (192.168.102.74:3210)
공격자	Kali Linux (192.168.216.131)

■ 취약점 테스트

Step 1. 환경 구성

CVE-2024-47066 취약점 테스트를 위한 docker 환경이 저장된 EQST Lab 의 GitHub Repository URL 은 다음과 같다.

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

우선, 피해자 PC 에서 git clone 명령어를 사용하여 CVE-2024-47066 저장소의 파일을 다운로드한다. 이후 다음 커맨드를 입력하여 간단하게 환경구축을 할 수 있다.

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

해당 환경의 경우 1.19.12 버전을 사용중이기 때문에 취약한 환경임을 확인할 수 있다.

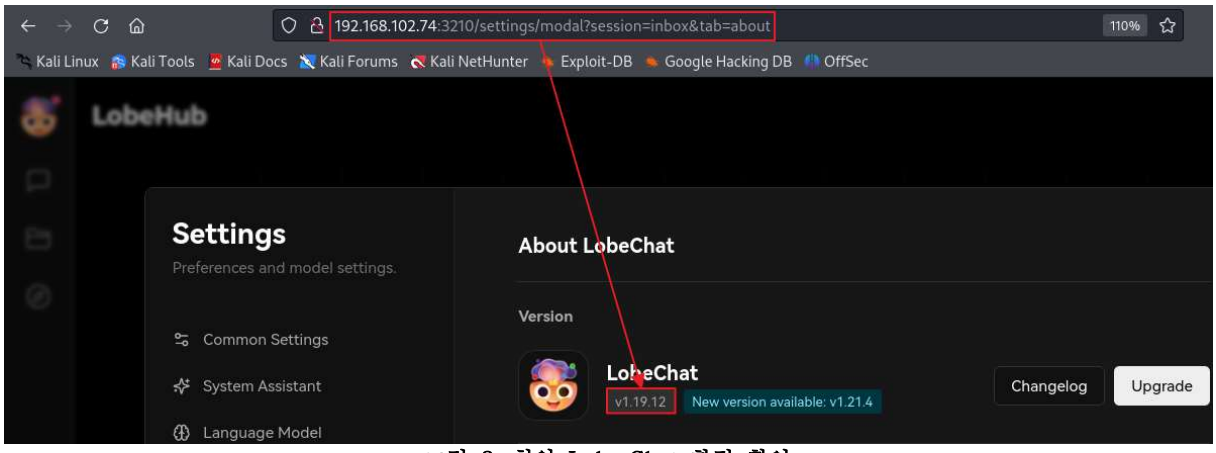


그림 3. 취약 Lobe Chat 환경 확인

Step 2. 취약점 테스트

CVE-2024-47066 취약점 테스트를 위한 PoC 가 저장된 EQST Lab 의 GitHub Repository URL 은 다음과 같다.

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

공격자 PC 에서 git clone 명령어를 사용하여 CVE-2024-47066 저장소의 PoC 를 다운로드한다.

```
┌──(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.
```

그림 4. CVE-2024-47066 PoC 다운로드

PoC 파일은 CVE-2024-47066.py 로 실행할 수 있고, 공격자 PC 에서 전송한 페이로드가 피해자의 Lobe Chat 에서 실행된다.

```
$ python3 CVE-2024-47066.py –v [Lobe Chat 페이지] –i [내부 페이지]
```

해당 환경은 내부에서만 접근 가능한 http://www.internal-service:4000 주소가 구축되어 있고, 해당 서비스에 SSRF 공격을 시도하는 커맨드는 다음과 같다.

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

해당 PoC 실행 커맨드를 아래와 같이 공격자 PC 에서 입력한다.

```
┌──(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
```

그림 5. PoC 실행 커맨드 예시

이후 피해자 PC 가 내부에 구축된 환경인 http://www.interal-service:4000 을 불러오는 것을 확인할 수 있다.

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

그림 6. 내부 데이터 탈취 예시

■ 취약점 상세 분석

취약점 상세 분석에서는 CVE-2024-47066 취약점이 발생하는 원리와 공격 시나리오를 순차적으로 설명한다. Step 1 에서는 사용자로부터 입력 받은 주소 검증 로직과 이를 우회하는 방안을 다룬다. Step 2 에서는 SSRF 공격에 대해 설명하고, Lobe Chat 에 가할 수 있는 공격 시나리오를 설명한다.

Step 1. SSRF 공격에 취약한 지점 탐색

1) 사용자 설정 플러그인 (Custom Plugin)

다양한 플러그인을 지원하는 Lobe Chat 은 Plugin Store 내 플러그인 외에도 사용자 설정 플러그인을 지원한다.

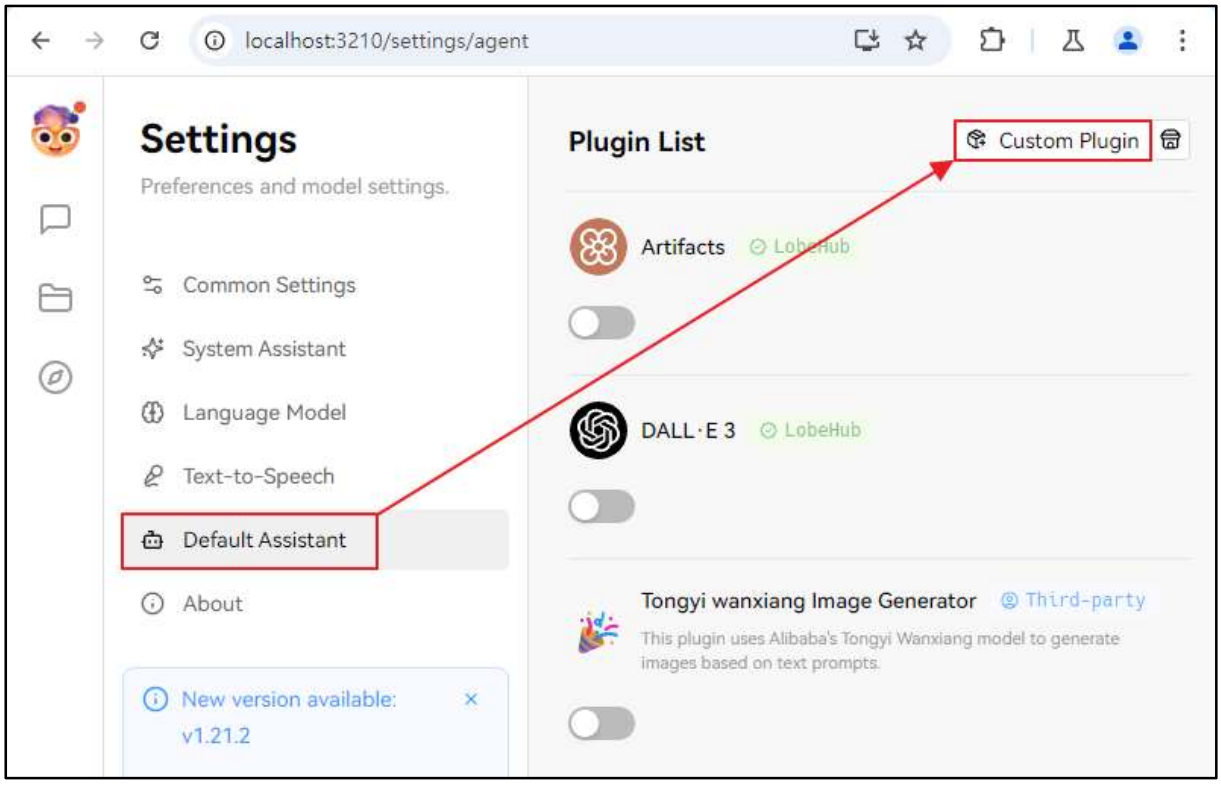


그림 7. Custom Plugin 접근 경로

이를 불러올 때는 플러그인 기능이 구현되는 방법을 기술한 Manifest 파일의 주소를 입력해야 한다. Manifest 파일은 아래와 같은 사항들을 포함한다.

구분	설명
identifier	플러그인 고유 식별자
api	플러그인의 모든 API 인터페이스가 기재된 배열
ui	플러그인이 프런트엔드 인터페이스를 불러오는 주소
gateway	API 인터페이스를 질의하기 위한 게이트웨이 지정
version	플러그인의 버전 정보

위 명세에 따라 Manifest 파일은 다음과 같은 json 파일로 구성된다.

```
{
    "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"
}
```

Lobe Chat 에서 사용자 설정 플러그인을 불러올 때 Same-Origin Policy[5]에 위배되어 에러가 나타날 수 있다. 이는 proxy 를 활용해 해결할 수 있도록 구성되어 있다.

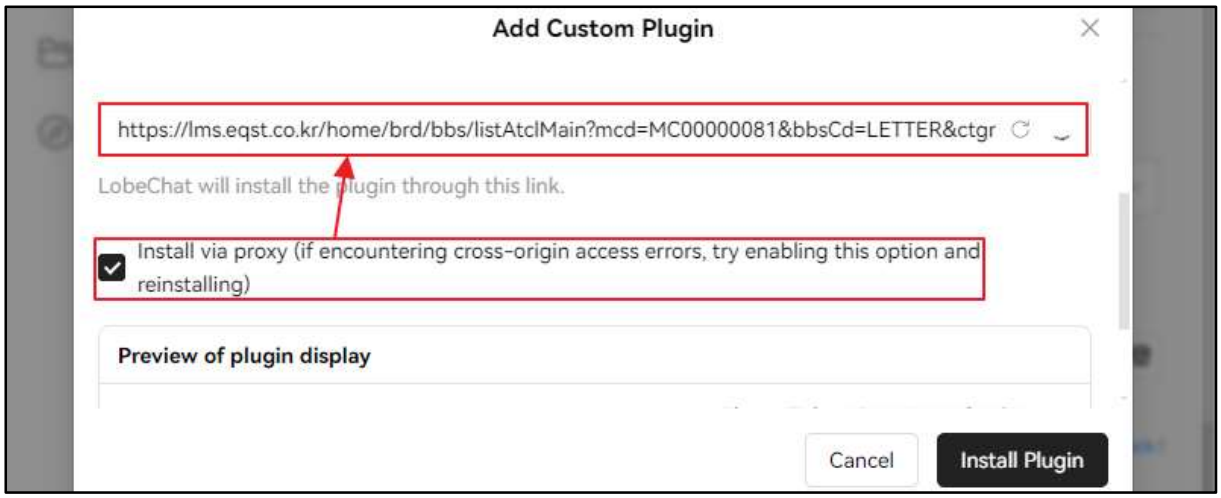


그림 8. 사용자 설정 플러그인 불러오기

proxy 를 활용한 요청은 다음과 같이 /api/proxy 엔드포인트에서 지정한 경로에 요청을 보내 응답을 불러온다.

[5] Same-Origin Policy (SOP): 어떤 출처에서 불러온 문서나 스크립트가 다른 출처에서 가져온 리소스와 상호 작용하는 것을 제한하는 보안 매커니즘

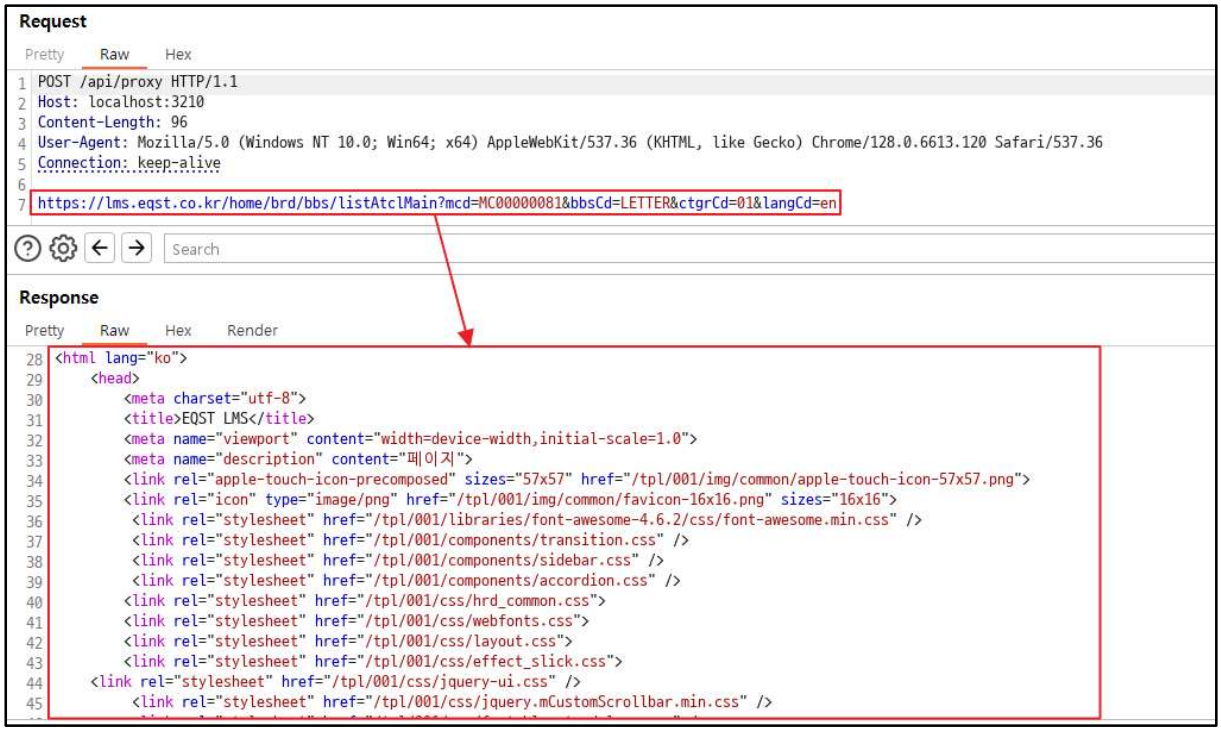


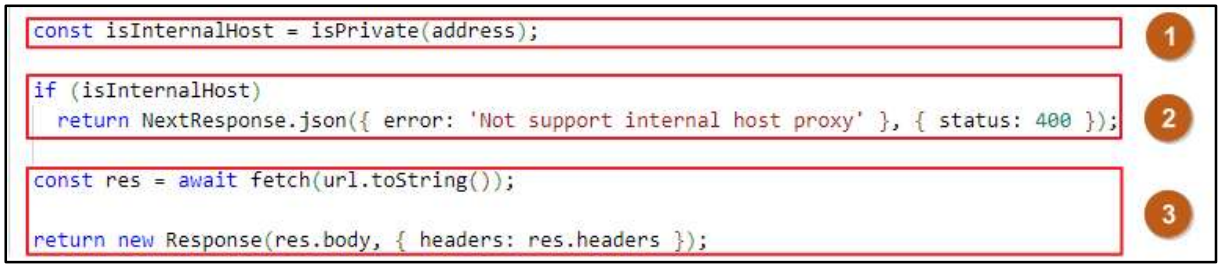
그림 9. /api/proxy 요청 결과

2) /api/proxy 엔드포인트 분석

/api/proxy 는 다음과 같은 TypeScript 코드로 구성되어 있다.

```typescript
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 });
};
```

위 코드에 따라 주소 검증 과정은 다음을 거친다.



① address 에 저장된 페이지의 IP 주소를 ip 모듈의 isPrivate 을 통해 내부망 주소인지 확인한 뒤 결과를 isInternalHost 에 저장한다.

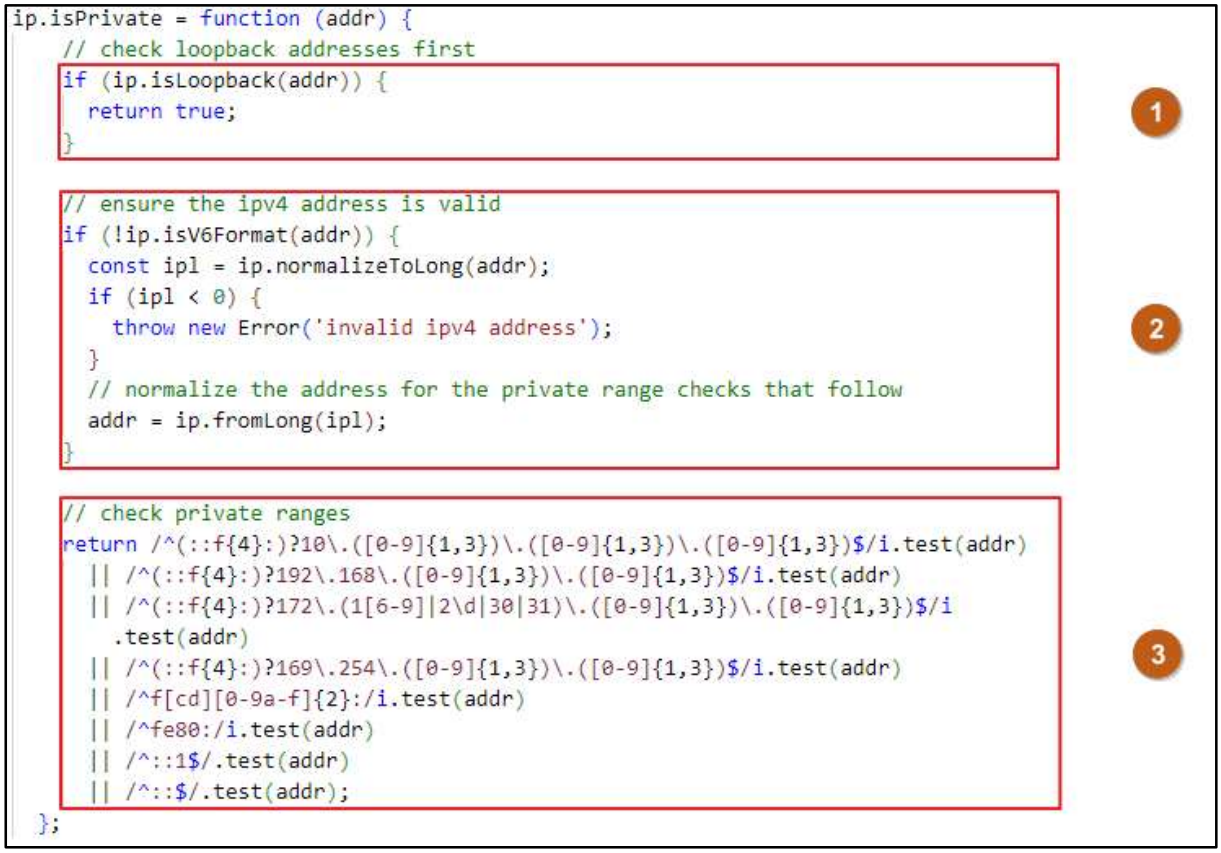
② isInternalHost 값이 true 라면 요청을 보내지 않고 400 에러를 반환한다.

③ isInternalHost 값이 false 라면 fetch 로 요청을 보내고 응답을 반환한다.

isPrivate 함수는 'ip' 모듈 내 구현되어 있다. 해당 모듈은 다음 GitHub Repository 에서 소스코드를 확인할 수 있다.

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

해당 Repository 내 lib/ip.js 파일을 살펴보면 다음과 같이 isPrivate 검증이 구현된 것을 확인할 수 있다.



① 자기자신을 가리키는 IP 인 루프백 주소인지 확인한다.

② isV6Format 으로 주소가 IPv6 형식인지 확인하며, 아닐 경우 IP 주소를 숫자로 변환해서 음수가 아닌지 확인한다.

③ 위 과정을 모두 통과했다면, 사설 IP 주소 범위인지 정규표현식으로 확인한다.

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[6])
fc00::/7, fd00::/8 (사설 IPv6)
fe80::/10 (IPv6 Link Local Address)
::1, :: (루프백 주소)

3) /api/proxy 엔드포인트 필터링 우회

위 사항을 고려했을 때, IP 주소를 기반으로 필터링하며 이외 필터링 로직은 없음을 알 수 있다. JavaScript 의 fetch 함수는 요청 시 다음과 같은 기본 옵션 값을 갖는다.

```
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 Referer 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
  });
```

이 중 redirect 옵션은 요청하는 URL 의 리다이렉션을 따라갈지 결정한다. follow 키 값은 자동으로 리다이렉션되는 URL 에 요청을 하고, manual 키 값은 리다이렉션을 따르지 않는다. error 키 값은 리다이렉션 발생 시 에러를 반환한다. redirect 설정의 기본 키 값은 follow 이므로 리다이렉션이 응답으로 올 경우 요청을 보내고 그 응답을 받게 된다. 내부 네트워크의 IP 를 가지지 않는 임의의 주소가 내부 네트워크의 IP 주소로 리다이렉션 응답을 반환한다면, 위에서 설명한 fetch 함수의 특징 때문에 내부 네트워크로 요청을 보내고 응답을 받을 수 있다.

[6] 링크 로컬 주소(Link Local Address): 단일 링크 내로 범위가 제한된 IPv6 유니캐스트 주소

Step 2. SSRF 공격

1) SSRF(Server-Side Request Forgery) 공격

Server-Side Request Forgery(SSRF) 공격은 공격자가 서버 측 애플리케이션이 의도하지 않은 위치로 요청을 보내도록 유도하는 웹 취약점이다. 이 공격을 통해 공격자는 서버가 내부 조직 인프라 서비스와 통신하도록 조작할 수 있다.

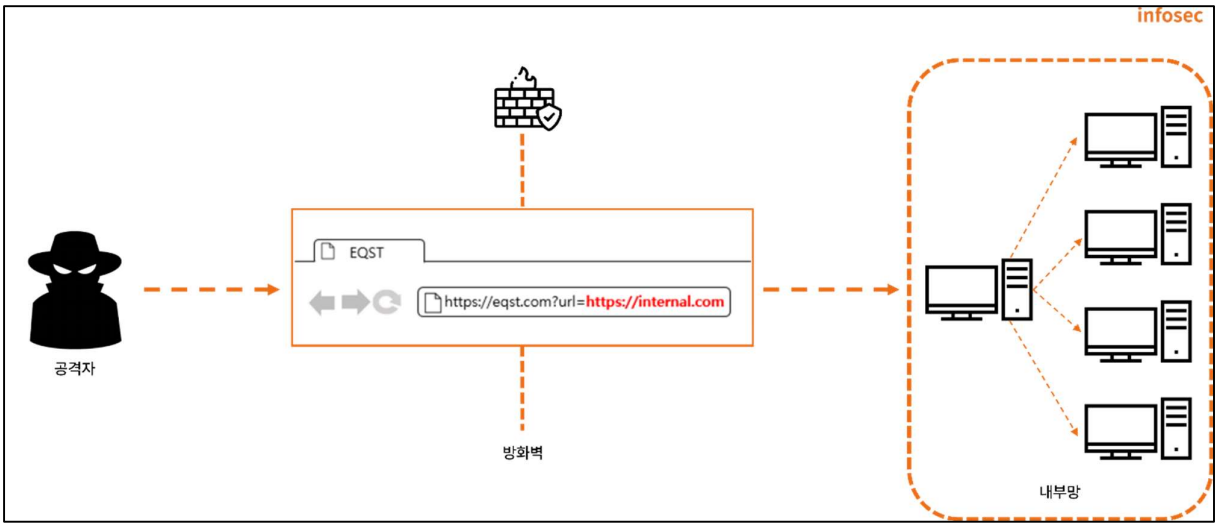


그림 10. SSRF 공격 개요

공격자는 해당 취약점을 통해 내부에서만 접근할 수 있는 민감 데이터에 접근할 수 있고, 일부 상황에서는 공격자가 SSRF 취약점을 이용하여 임의의 명령 실행을 수행할 수 있다. 또한, SSRF 취약점을 이용하여 제 3 자(3rd-party)에 대한 악의적인 공격을 시도한다면 해당 공격은 취약점이 존재하는 애플리케이션을 호스팅하는 서버에서 시작된 공격으로 볼 수 있다.

2) Lobe Chat SSRF 공격 시나리오

1. AI 정보 탈취

해당 AI 정보 탈취 시나리오는 다음과 같은 내부 LLM 환경에서 진행되었다.

오픈소스	주소
Ollama	192.168.102.231:16728

Ollama 의 경우 별도 인증 과정이 존재하지 않으므로, 내부 모델에 SSRF 취약점이 존재할 경우 RESTful API 로 LLM 정보 탈취가 가능하다.

Ollama 로 구축된 LLM 모델 주소에 요청을 보낼 경우, 다음과 같이 Ollama 가 동작 중인 것을 확인할 수 있다.

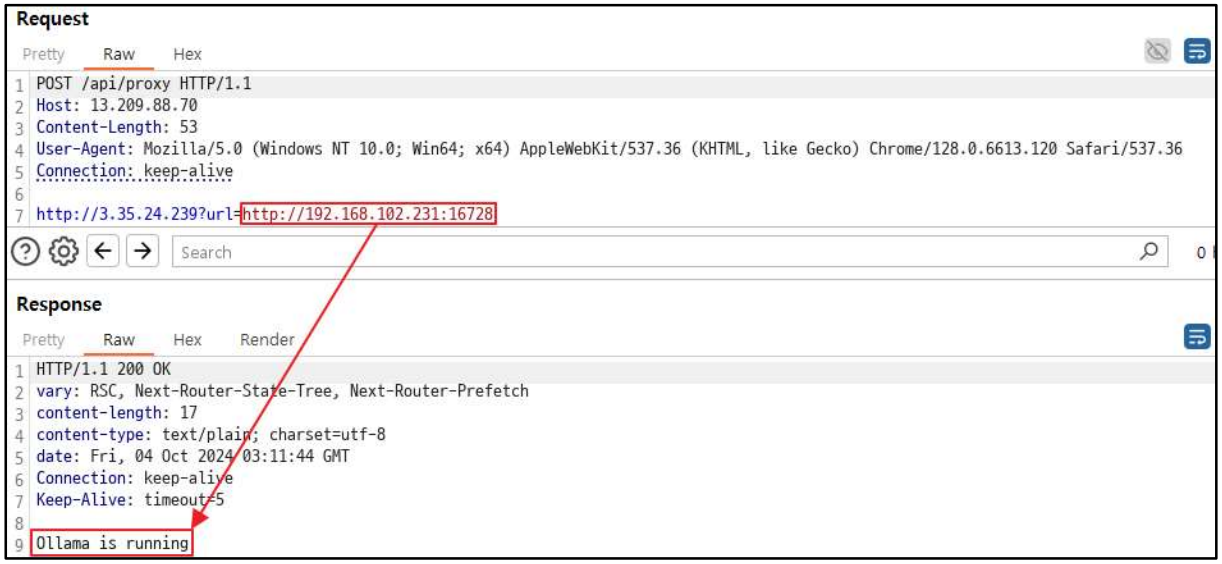


그림 11. LLM 모델 주소 접근 확인

해당 Ollama 에서 어떤 LLM 모델이 사용되는지 /api/tags 경로에 요청을 보내 확인할 수 있다.

그림 12. /api/tags 접근

/api/ps 경로에 요청을 보내면 어떠한 LLM 모델이 메모리에 올라가 있는지도 확인 가능하다.

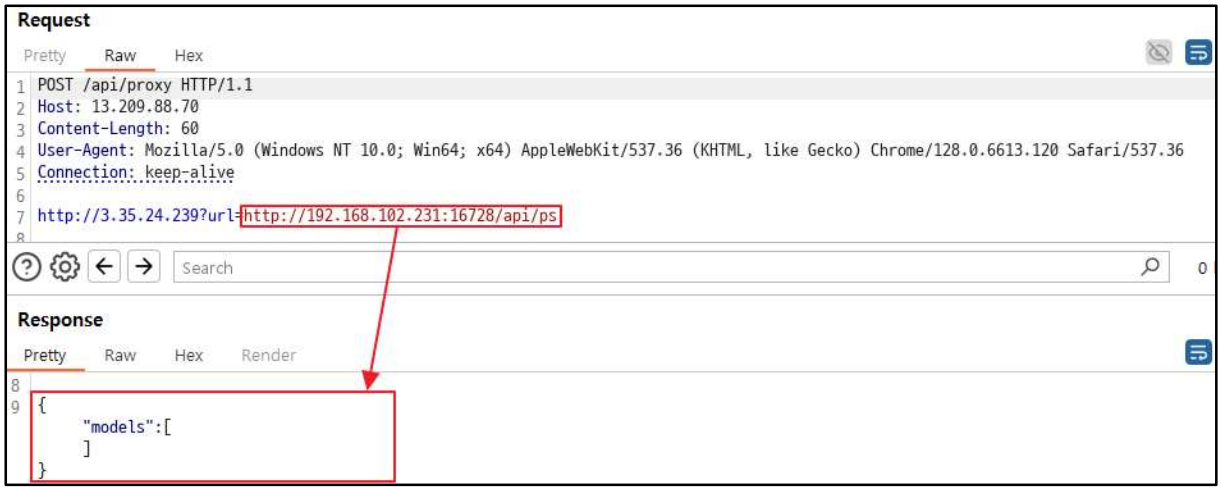


그림 13. /api/ps 접근

2. 클라우드 서비스 침투

역할	주소
피해자	3.35.156.32
공격자	3.35.24.239

클라우드 서비스 침투 시나리오를 가정하기 위해 피해자는 AWS 에서 서비스 중인 취약한 버전의 Lobe Chat 으로 구성한다. AWS 환경에서 인스턴스를 구성 또는 관리하는데 사용될 수 있는 관련 데이터를 메타데이터라고 칭한다. 이는 http://169.254.159.254 주소를 통해 접근할 수 있고, 인스턴스에서 IAM 임시 자격증명까지 접근 가능하기 때문에 해당 자격증명 탈취를 통한 인스턴스 통제까지 가능하다.

Metadata version1 을 사용하는 환경에서 다음과 같이 http://169.254.169.254/latest/meta-data 경로에 접근하면, 어떠한 메타데이터가 인스턴스 내 존재하는지 확인 후 불러올 수 있다. 해당 경로 내 iam/이 존재하는 점으로 해당 인스턴스가 IAM Role 을 사용하고 있다는 것을 추측할 수 있다.

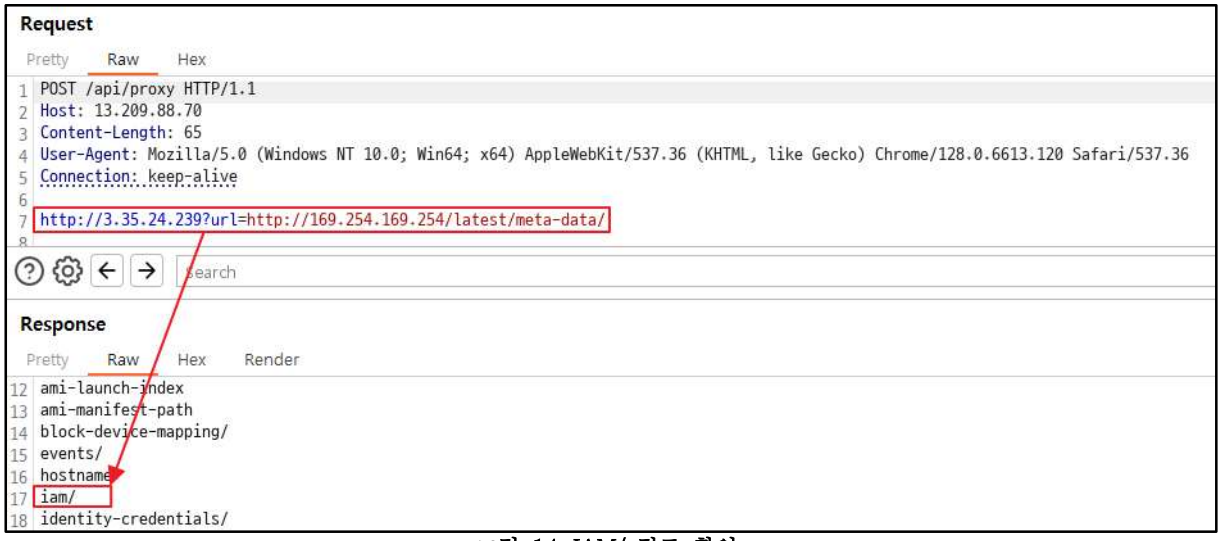


그림 14. IAM/ 경로 확인

이 중 iam/security-credentials 경로에는 rnt-ssrf 라는 IAM Role 이 저장되어 있다.

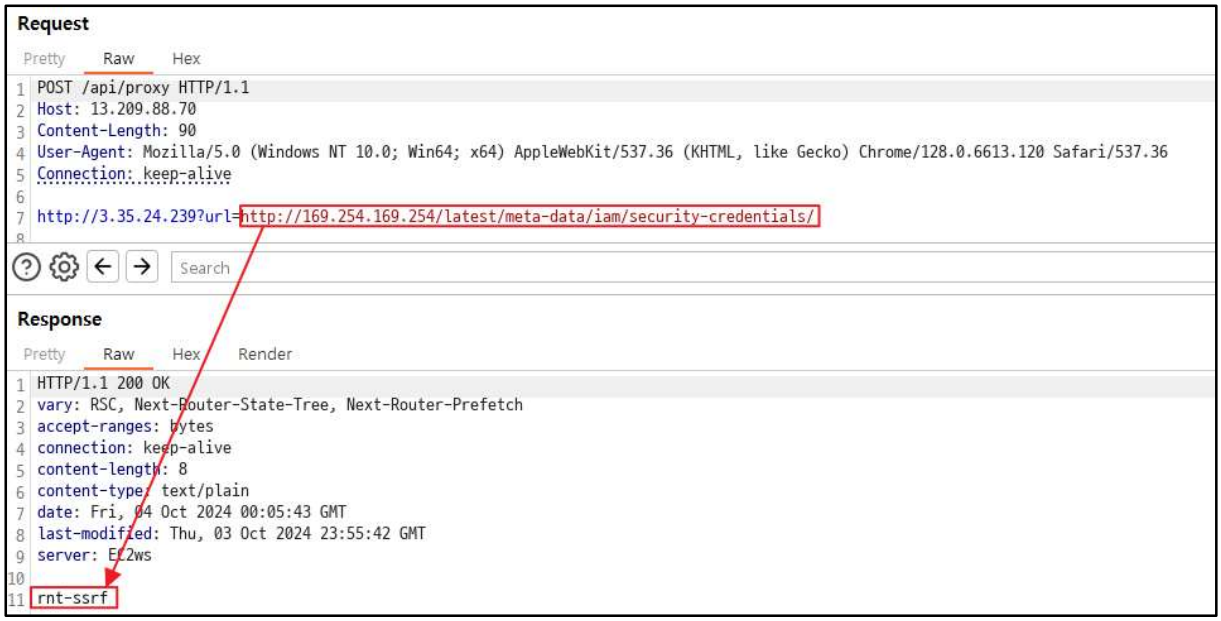


그림 15. IAM 자격 증명 경로 확인

해당 이름의 IAM Role 에 접근하면 다음과 같이 IAM 자격증명 정보를 획득할 수 있다.

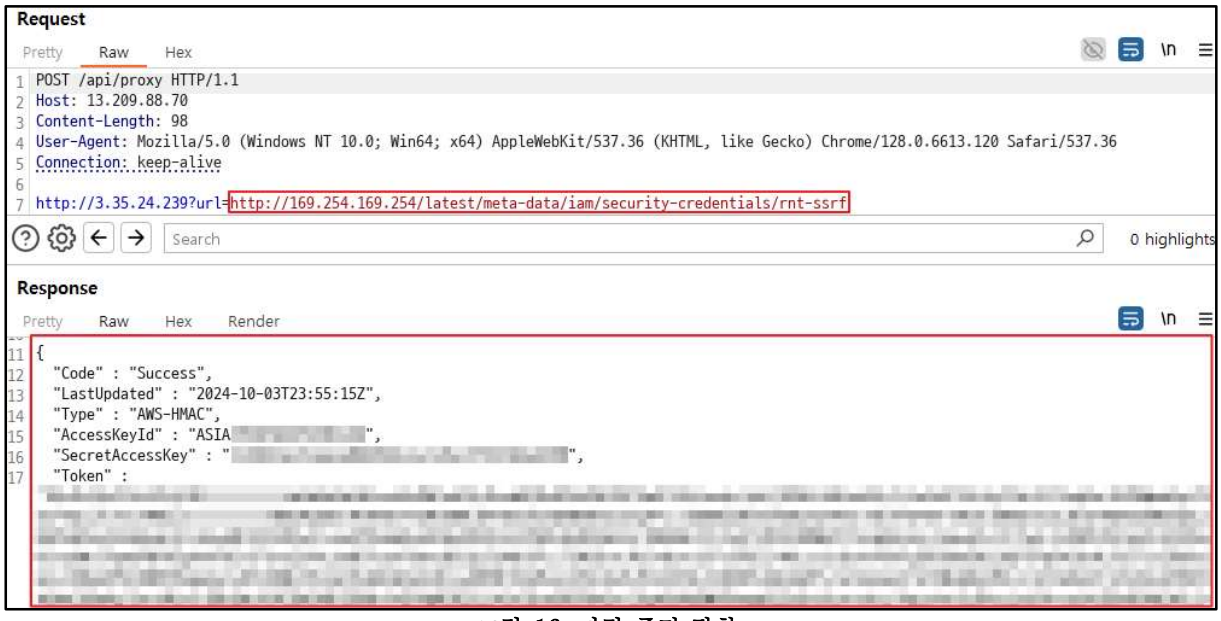


그림 16. 자격 증명 탈취

이후 IAM 정책 설정에 따라, 다음과 같이 ec2 인스턴스 제어권을 획득할 수 있다.

그림 17. 탈취한 자격 증명으로 인스턴스 침투

■ 대응 방안

CVE-2024-47066 가 발표되기 전인 9 월 20 일, 해당 취약점을 패치한 버전 1.19.13 이 출시되었다. 해당 버전 소스 코드는 아래 링크를 통해 확인할 수 있다.
•URL: https://github.com/lobehub/lobe-chat/tree/v1.19.13

해당 취약점 패치 내용은 아래 링크를 통해 확인할 수 있다.
•URL: https://github.com/lobehub/lobe-chat/commit/e960a23b0c69a5762eb27d776d33dac443058faf#diff-7863de9f92a2b10e6b7e0438075c9d9f2639640eb5310505c64a0da11add43f3R7

언급한 바와 같이 해당 패치에서 route.ts 의 위치와 코드 변경이 있는 것을 확인할 수 있다.

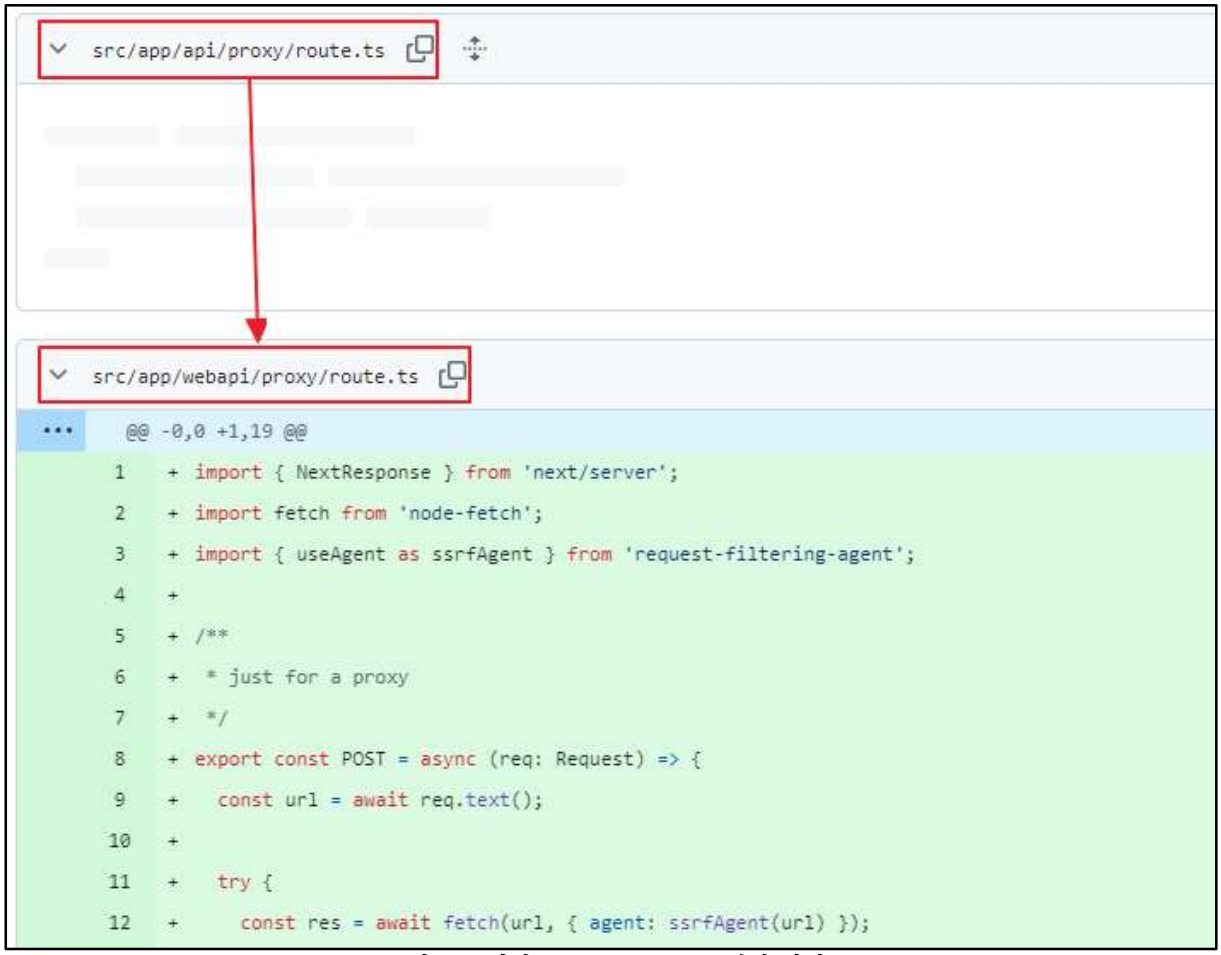


그림 18. 버전 1.19.13 route.ts 변경 사항

취약점이 발생했던 app/api/proxy/route.ts 코드를 app/webapi/proxy/route.ts 로 변경하였고, 해당 코드는 다음과 같다.

```
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 });
  }
};
```

패치 이후 SSRF 공격 방지 로직이 구현된 request-filtering-agent 모듈을 활용하여 fetch 함수를 통해 요청을 보내는 것을 확인할 수 있다.

해당 취약점 패치 작업은 다음과 같이 설정창에서 수행 가능하다.

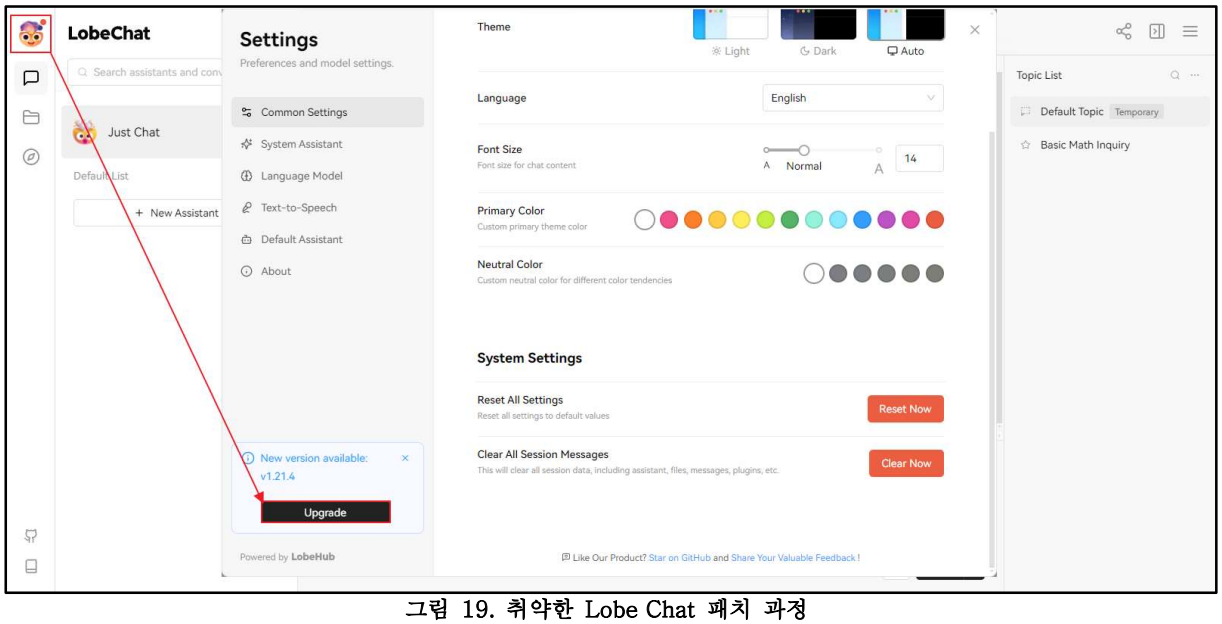


그림 19. 취약한 Lobe Chat 패치 과정

상세 패치 내역은 아래 링크에서 확인할 수 있다..

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

따라서, SSRF 취약점이 존재하여 공격이 가능한 1.19.12 버전 이하 취약한 버전의 Lobe Chat 사용자는 위 작업 과정에 따라 패치를 수행하여야 한다.

■ 참고 사이트

• 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
• 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