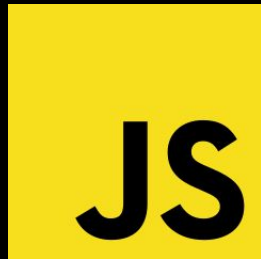


Staying Ahead of the Game

JavaScript Security

 / @dheerajhere



About Me

 Senior Frontend Engineer @ **GitLab**

 Ambidextrous TT Player

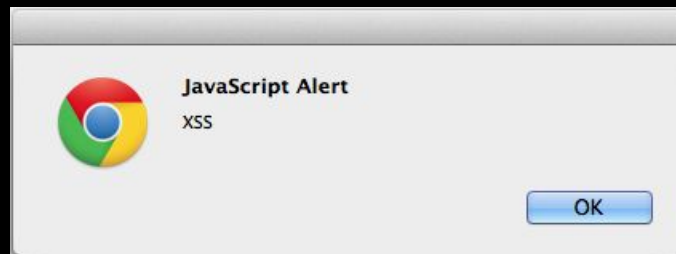
 I like to find Security bugs for fun and swags



This makes me happy!



Uber, Symantec, CKEditor, Dropbox,
Jenkins, MailChimp, Recruiterbox,
InVision, Intuit, etc.



Agenda

- Why?
- Understanding XSS
- What's CSP?
- Demo
- Initiatives at GitLab
- Best Practices
- Q&A

Security is important!



Why should we care about frontend Security?

One of the things we assume is that
Security is only on the Backend.

Story Time!

Stealing passwords from McDonald's users

Reflected XSS through AngularJS sandbox bypass causes password exposure of McDonald users.

By abusing an insecure cryptographic storage vulnerability ([link](#)) and a reflected server cross-site-scripting vulnerability ([link](#)) it is possible to steal and decrypt the password from a McDonald's user. Besides that, other personal details like the user's name, address & contact details can be stolen too.

Proof of Concept

Reflected XSS through AngularJS sandbox escape

McDonalds.com contains a search page which reflects the value of the search parameter (q) in the source of the page. So when we search on for example *****-test-reflected-test-*****, the response will look like this:

NEARLY 2000 WORDPRESS WEBSITES INFECTED WITH A KEYLOGGER



 Markdown Preview Security Risk 

 Can you preview the following Markdown?

TEST
[\[CLICK\]\(javascript:alert\(document.domain\)\)](#)


 Sure, here's a preview of the Markdown you provided:

less  Copy code

```
# TEST  
\[CLICK\]\(javascript:alert\(document.domain\)\)
```


And here's how it will look like when rendered:

TEST

[CLICK](#)

Please note that the ``javascript:`` code in the link could potentially execute arbitrary code in the user's browser and is considered a security risk. Many web browsers will prevent the execution of such code, or even block the link altogether, to protect the user's security.

ChatGPT Mar 14 Version: Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

javascript:alert(document.domain)

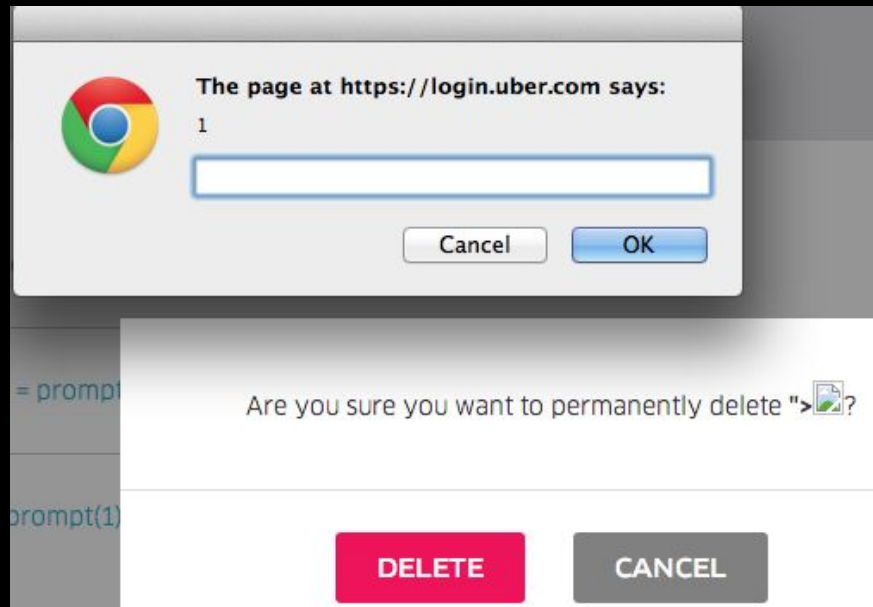
Let's talk XSS!



CROSS SITE SCRIPTING (XSS)

WHEN DATA BECOMES CODE

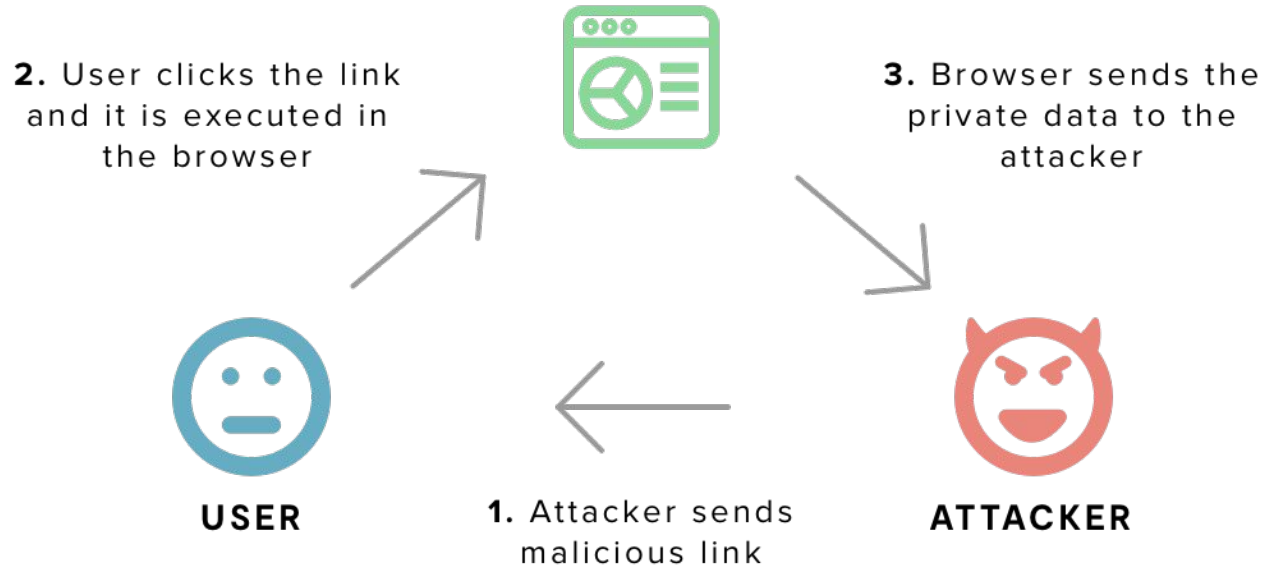
```
"><img src=x onerror=prompt(1)>
```



CROSS SITE SCRIPTING (XSS)

- It's more than the alert popup
- XSS attack users
- Hijack, Steal, Record

Typical Reflected XSS



DOM XSS



```
<html>
```

```
(...)
```

```
Dashboard for
```

```
<script>
```

```
    const pos = document.URL.indexOf("name=") + 5;
```

```
    const name = decodeURIComponent(document.URL.substring(pos));
```

```
    document.write(name);
```

```
</script>
```

```
(...)
```

```
</html>
```

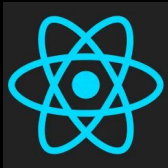
Mitigations...

- Analyze places where **DOM** elements are created
- Avoid **innerHTML** and similar functions
- Add a **linter** rule to prevent its usage
- Input sanitization
- Output Encoding

Enable HTTPOnly, Secure flag for sensitive cookies

```
app.use("/cookies", (req, res) => {  
  const dataToSecure = {  
    dataToSecure: "This is the secret data in the cookie.",  
  };  
  
  res.cookie("secureCookie", JSON.stringify(dataToSecure), {  
    → secure: process.env.NODE_ENV !== "development",  
    → httpOnly: true,  
    expires: dayjs().add(30, "days").toDate(),  
  });  
  
  res.send("Hello.");  
});
```

Demo

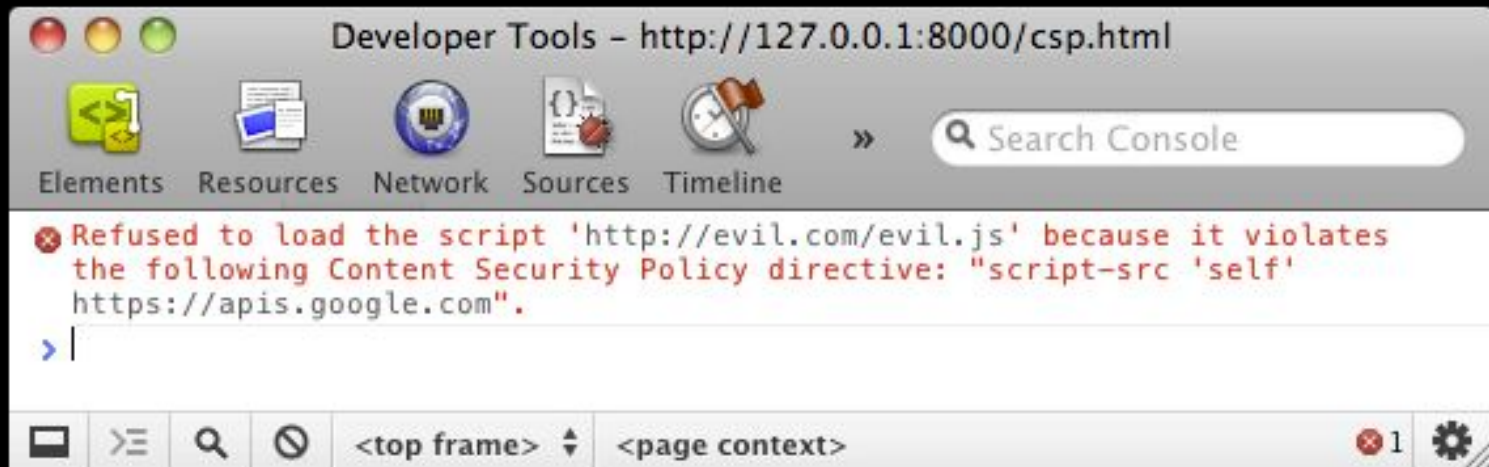


Rule of Thumb



- Never trust user input
- Avoid using `dangerouslySetInnerHTML`, `v-html`
- Sanitize input, escape output
- Build secure `links`
- Add `eslint` to prevent improper usages

Content Security Policy (CSP)



Content Security Policy

- Mitigating Cross-site Scripting Attacks
- Secure Form Submission
- Mitigating Clickjacking
- HTTPS

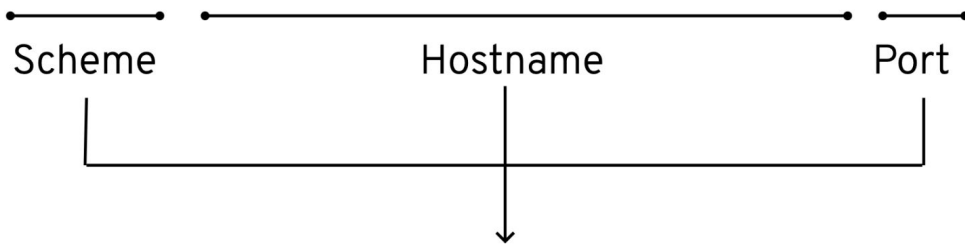
Add CSP header

```
app.use(  
  helmet({  
    contentSecurityPolicy: {  
      useDefaults: false,  
      directives: {  
        defaultSrc: ["'self'"],  
        scriptSrc: ["'self'", "example.com"],  
        objectSrc: ["'none'"],  
        upgradeInsecureRequests: [],  
      },  
    },  
  })  
);
```

Same-origin Policy

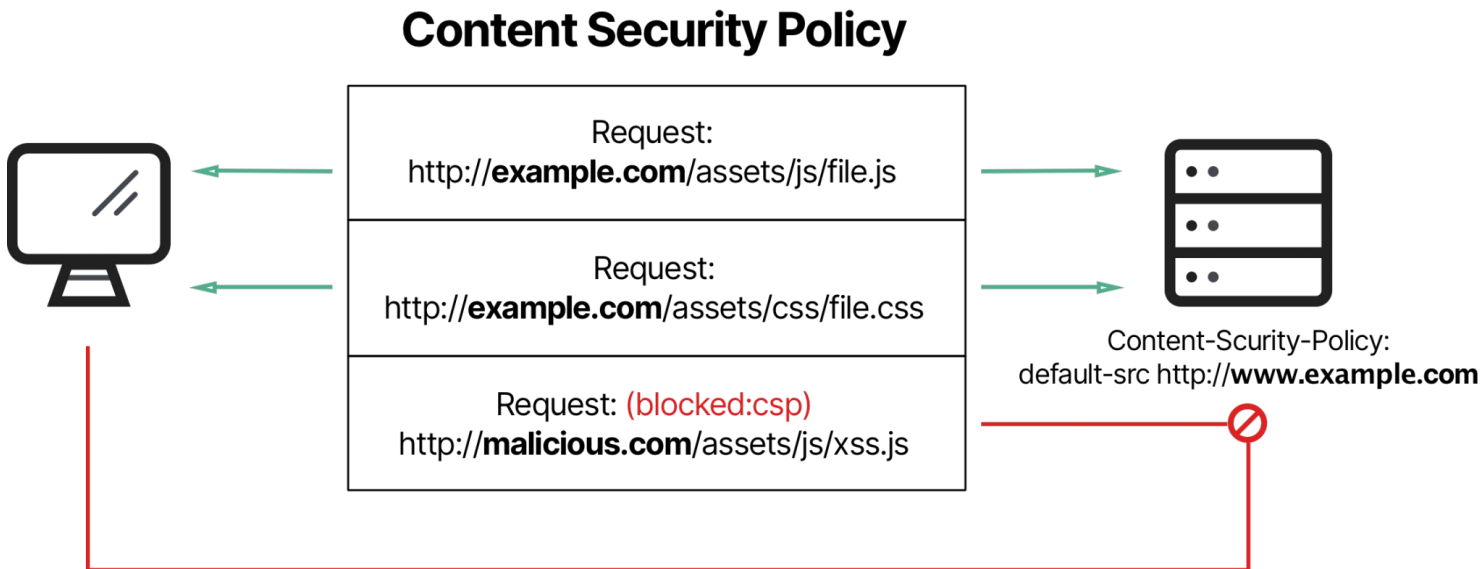
'SELF'

http:// www.cspisawesome.com :80



ORIGIN

CSP blocks malicious requests



Demo?

Other HTTP Security headers

- **X-Frame-Options:** deny
- **Strict-Transport-Security:**
max-age=16070400; includeSubDomains
- **X-XSS-Protection:** 1; mode=block
- **X-Content-Type-Options:** nosniff



Securing GitLab's Frontend

Everything at GitLab starts with an issue



[GitLab.org](#) > [GitLab](#) > [Issues](#) > **#219124**

Open

Issue created 2 years ago by **Dheeraj Joshi** Developer

Close issue

Improve Frontend Security Posture

This issue should list down all the possible areas where we can make improvement to strengthen our Frontend Security.

Mitigating Cross-site Scripting (XSS)

Update Sanitizer

1. Swap `sanitize-html` for `dompurify` (more robust) - [!31928 \(merged\)](#), [gitlab-ui!1636 \(merged\)](#)

Avoid v-html

Since we know, `v-html` is bad

1. Add `v-safe-html` directive which sanitizes html by default - [gitlab-ui!1413 \(merged\)](#)
2. Add ESLint rules to prevent using `v-html` - [#232488 \(closed\)](#)
3. Audit and remove existing `v-html` usages - [&4273 \(closed\)](#)

Prevent URL injection

1. Add `safe-link` directive to prevent url injection - [Documentation](#), [gitlab-ui!1457 \(merged\)](#)
2. `GILink` component should prevent JS execution by default - [gitlab-ui!1472 \(merged\)](#)
3. `GILButton` component should prevent JS execution by default - [gitlab-ui#1379 \(closed\)](#)

Development guidelines

1. Frontend Security Best Practices - https://docs.gitlab.com/ee/development/secure_coding_guidelines.html#xss-guidelines

<https://gitlab.com/gitlab-org/gitlab/-/issues/219124>



As we know, v-html is bad

→ We built **v-safe-html**

Directive	Output
v-html	Hello <code><script>alert(document.domain)</script></code> world!
v-safe-html	Hello world!

<https://gitlab-org.gitlab.io/gitlab-ui/?path=/story/directives-safe-html-directive--default>



We also made our **Link component** safe-by-default



```
<gl-link href="javascript:alert(1)">click me</gl-link>
```



```
<a href="about:blank" class="gl-link">click me</a>
```

<https://gitlab-org.gitlab.io/gitlab-ui/?path=/story/base-link--default-link>

Iframe Sandboxing

▸ We use a third-party module to generate charts!

▸ It has caused numerous XSSes in the past and \$\$\$\$!

▸ We implemented sandboxing and fixed all the issues with one single change.

https://gitlab.com/gitlab-org/gitlab/-/merge_requests/74414



GitLab.org > GitLab > Merge requests > !74414

Render GFM Mermaid diagrams in a sandboxed iframe

Merged Dheeraj Joshi requested to merge `djadmin-sandbox-mermaid` into `master` 1 year ago

Overview 139 Commits 6 Pipelines 24 Changes 16

Related issues: [#342208](#) (closed), [#345592](#) (closed)

What does this MR do and why?

This MR moves rendering of mermaid diagrams in GFM within a sandboxed environment by using `iframe's sandbox attribute`. This should help in minimizing the impact from XSS vulnerabilities caused by Mermaid, and possibly with other 3rd party libraries in the future.

Details:

1. Each mermaid diagram is rendered in a separate iframe (`https://<gitlab-instance>/-/sandbox/mermaid`)
2. The iframe gets loaded in a cross-origin sandboxed environment
3. The iframe gets the diagram source via `postMessage` once it's loaded
4. The iframe draws the diagram and sends a `postMessage` to `parent` window communicated the rendered diagram size
5. Parent window receives the `postMessage` and adjusts the iframe size.
6. Act natural and say goodbye to `Found a new mermaid XSS vulnerability` emails

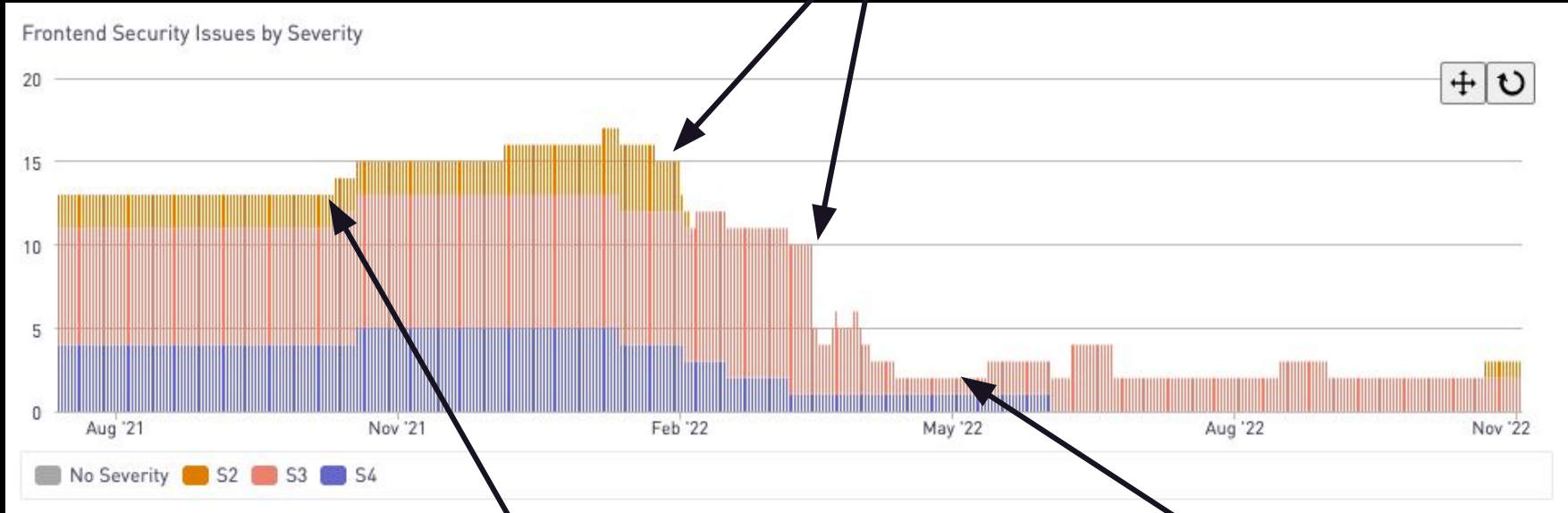
Note: The changes are behind a new `feature flag` `sandboxed_mermaid`.

Screenshots or screen recordings

▸ how a blocked XSS attack looks like

^ An example to illustrate that

Frontend Security Issues



Issues increase

Issues drop significantly

How to improve?

- Shift Left - Integrate into SDLC
- Adopt Secure-by-Design principles
- Use Security Middleware like [Helmet.js](#)
- Use standard Sanitizers like [DOMPurify](#)
- [Snyk](#), Npm audit

Learning Resources

1. Stanford - **CS 253** Web Security Course
2. **OWASP** Developer Guidelines
3. Hacker101
4. Play **CTFs**
5. GitLab's **Secure Coding Guidelines**

Thank you



 / @dheerajhere