Securing React with Trusted Types

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

```
Philippe De Ryck<img
src="none"
onerror="alert('OMG')">
```

```
<h1>
  Welcome Philippe De Ryck
  <img src="none"
  onerror="alert('OMG')">
</h1>
```

Multiple XSS vulnerabilities in child monitoring app Canopy ‘could risk location leak’

Jessica Haworth 06 October 2021 at 14:25 UTC
Updated: 07 October 2021 at 09:09 UTC

XSS Vulnerabilities Mobile

Pair of unpatched security bugs are ‘just the tip of the iceberg’

Facebook pays out $25k bug bounty for chained DOM-based XSS

Adam Bannister 09 November 2020 at 17:55 UTC
Updated: 11 November 2020 at 11:45 UTC

Bug Bounty | Social Media | XSS

Researcher awarded five-figure sum for 'easy to exploit' bug

Oculus, Facebook account takeovers net security researcher $30,000 bug bounty

Adam Bannister 05 January 2021 at 15:28 UTC
Updated: 06 January 2021 at 11:16 UTC

Facebook | Bug Bounty | XSS

XSS in virtual reality forum one of three flaws chained to land bumper payout

Trusted Types has the ability to eradicate DOM-based XSS in your entire application

I am *Dr. Philippe De Ryck*

Pragmatic Web Security
Security for developers

Founder of Pragmatic Web Security

Google Developers Experts

Google Developer Expert

AMBASSADOR PROGRAM

Auth0 Ambassador

Secure Application Development

SecAppDev organizer

I help developers with security

- **Hands-on in-depth security training**
- **Advanced online security courses**
- **Security advisory services**

https://pragmaticwebsecurity.com

```
Philippe De Ryck<img
src="none"
onerror="alert('OMG')">
```

A JSX template to combine data with HTML

```
1  return ( <h1> Welcome { name } </h1>);
```

React applies automatic escaping to data embedded in JSX templates

The data seen by the browser

```
1  Philippe De Ryck&lt;img src="none"
2  onerror="alert('OMG')"&gt;
```

The browser does not see HTML code, but simply renders the HTML tags

ee greatest things you learn from traveling

the great things on earth traveling teaches us by example. Here are some of the mo
ous lessons I've learned over the years of traveling.

Leaving your comfort zone might lead you to such beautiful sceneries like this one.

ppreciation of diversity

tting used to an entirely different culture can be challenging. While it's also nice to learn abo
es online or from books, nothing comes close to experiencing cultural diversity in perso
n to appreciate each and every single one of the differences while you become mo
fluid.

dangerouslySetInnerHTML

An example using dangerouslySetInnerHTML

```
1  return ( <div>
2    <h3>{ title }</h3>
3    <p dangerouslySetInnerHTML={{__html: review}}></p>
4  </div>);
```

DATA

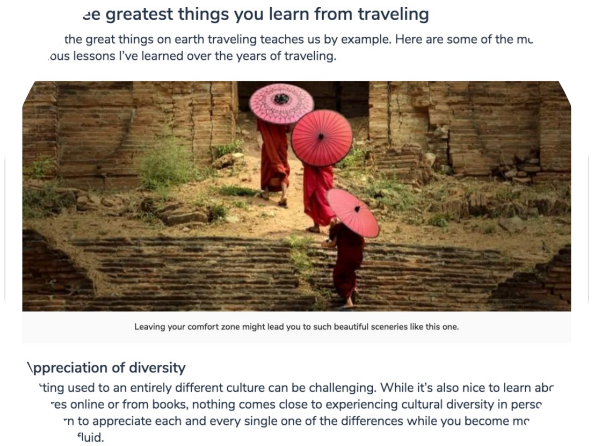
APPLICATION CODE

Application

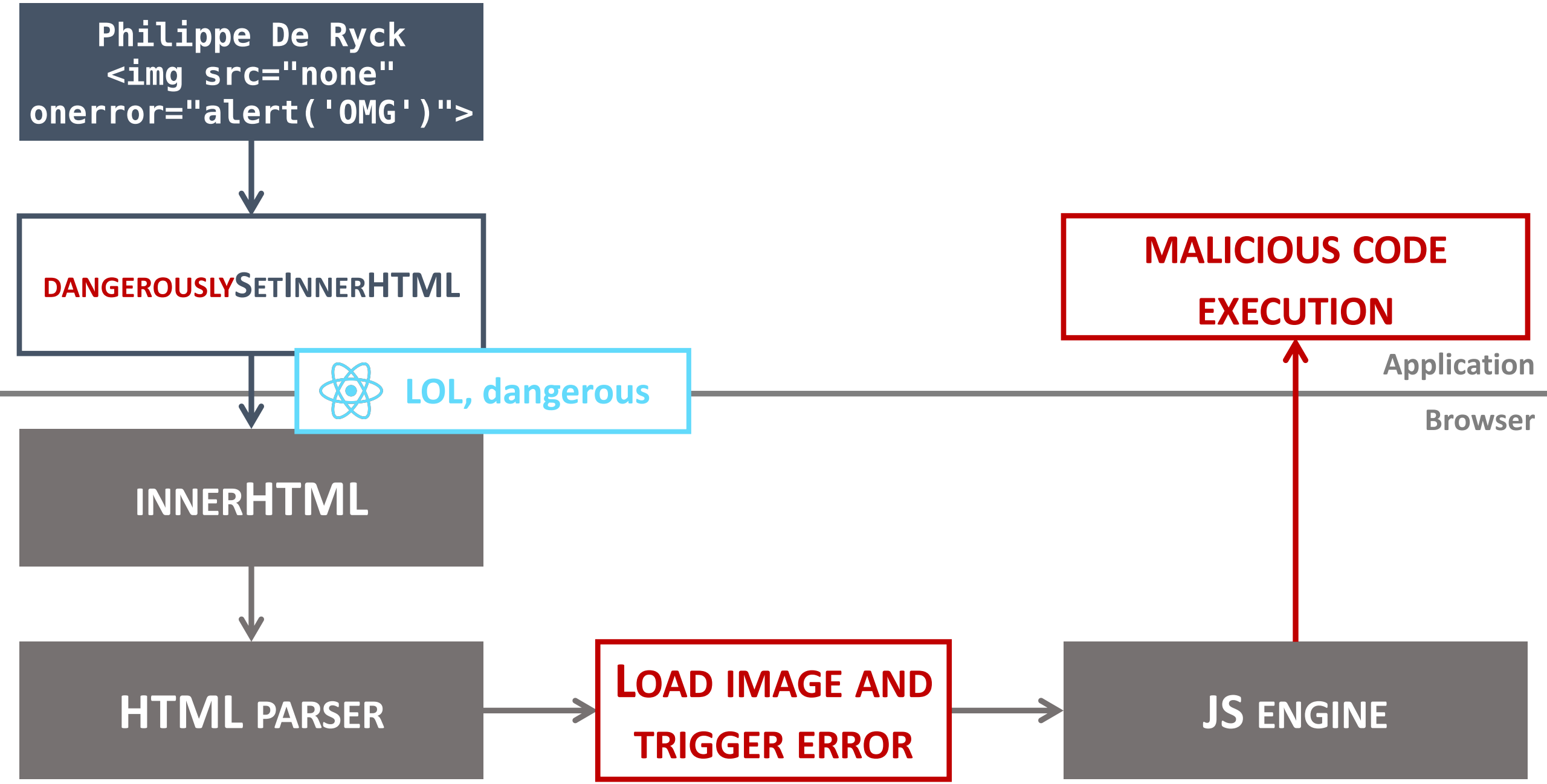
Browser

DANGEROUS SINKS (INNERHTML, ...)

HTML PARSER



@PhilippeDeRyck


Philippe De Ryck
<img src="none"
onerror="alert('OMG')">
DANGEROUSLYSETINNERHTML
LOL, dangerous
MALICIOUS CODE EXECUTION
Application
Browser
INNERHTML
HTML PARSER
LOAD IMAGE AND TRIGGER ERROR
JS ENGINE


DATA
escaping / sanitization
Application
Browser
DANGEROUS SINKS
(INNERHTML, ...)
HTML PARSER

A JSX template to render user-provided HTML with a major vulnerability

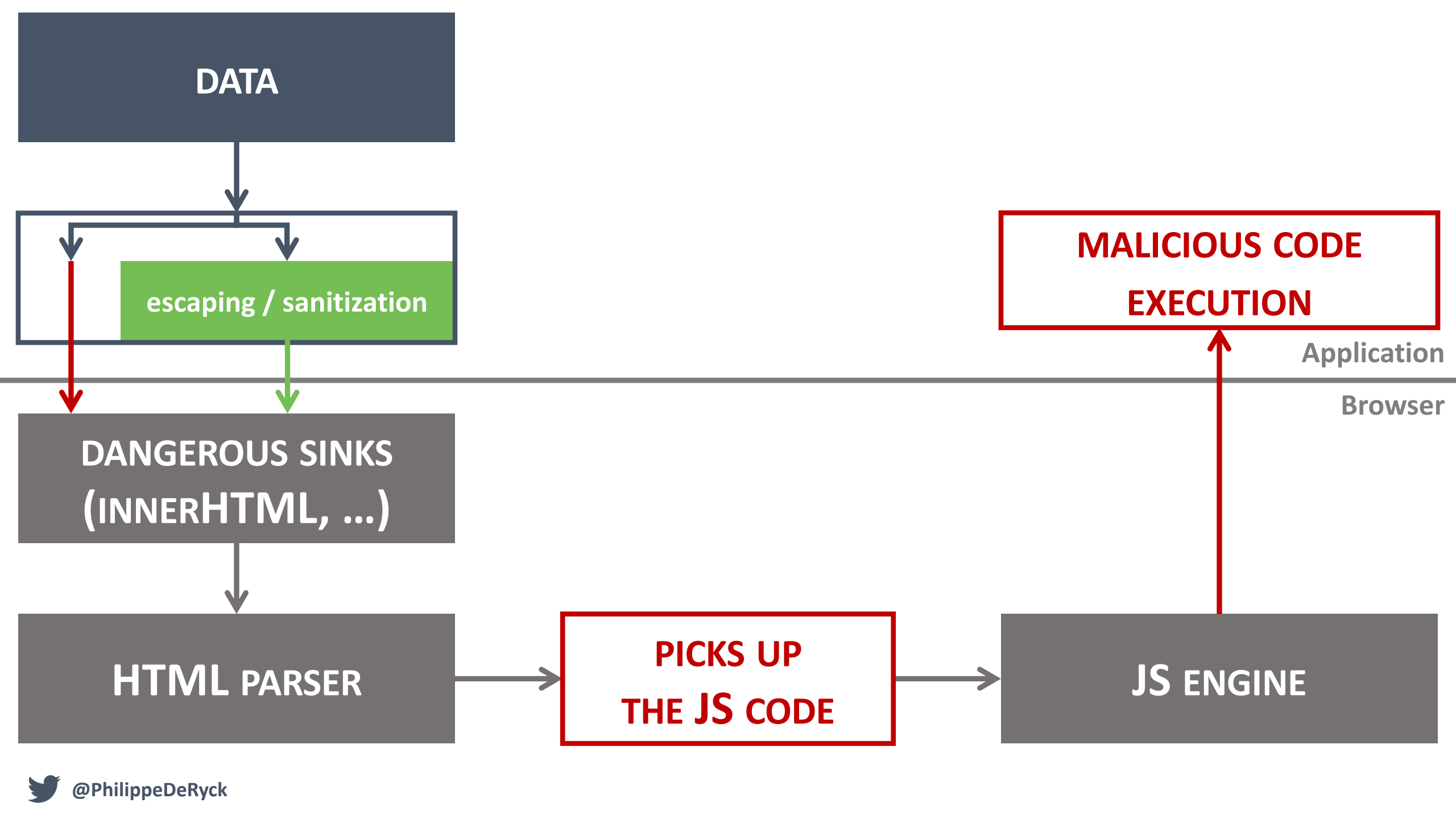
```
1  return ( <div>
2    <h3>{ title }</h3>
3    <p dangerouslySetInnerHTML={{__html: review}}></p>
4  </div>);
```

This property is dangerous, since React does not apply any protection at all

A JSX template to render user-provided HTML using DOMPurify

```
1  import DOMPurify from 'dompurify';
2
3  return ( <div>
4    <h3>{ title }</h3>
5    <p dangerouslySetInnerHTML={{__html: DOMPurify.sanitize(review)}}></p>
6  </div>);
```

DOMPurify turns untrusted HTML in safe HTML, making it safe to include in the page


DATA
escaping / sanitization
MALICIOUS CODE EXECUTION
Application
Browser
DANGEROUS SINKS (INNERHTML, ...)
HTML PARSER
PICKS UP THE JS CODE
JS ENGINE

Signal Messenger

```
@@ -111,7 +113,9 @@ export class Quote extends React.Component<Props, {}> {
```

```
111
112         if (text) {
113           return (
-114           <div className="text" dangerouslySetInnerHTML={{ __html: text }} />
115           );
116         }
117
```

```
113
114         if (text) {
115           return (
+116           <div className="text">
+117             <MessageBody text={text} />
+118           </div>
119           );
120         }
121
```


https://github.com/signalapp/Signal-Desktop/commit/4e5c8965ff72576a9e20850dd30d9985f4073192#diff-8bba204372da85d8cceed81278b7eec

Using Refs *as an escape hatch to access the raw DOM*

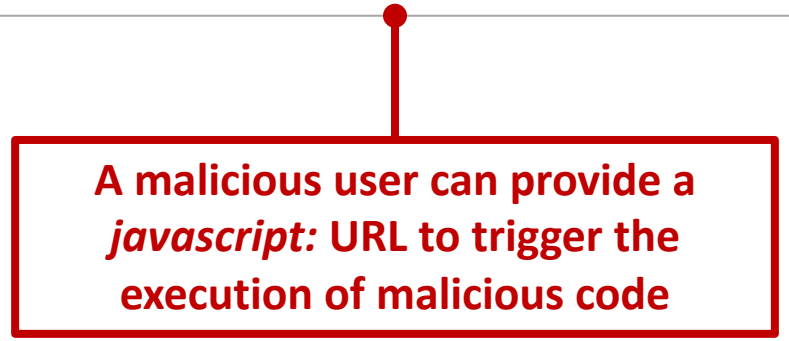
```
1  function App() {
2    const messageBoxRef = React.createRef();
3
4    useEffect(() => {
5      let messages = "...";
6      messageBoxRef.current.innerHTML += messages;
7    })
8
9    return (<div ref={messageBoxRef}>No new messages</div>);
10 }
```

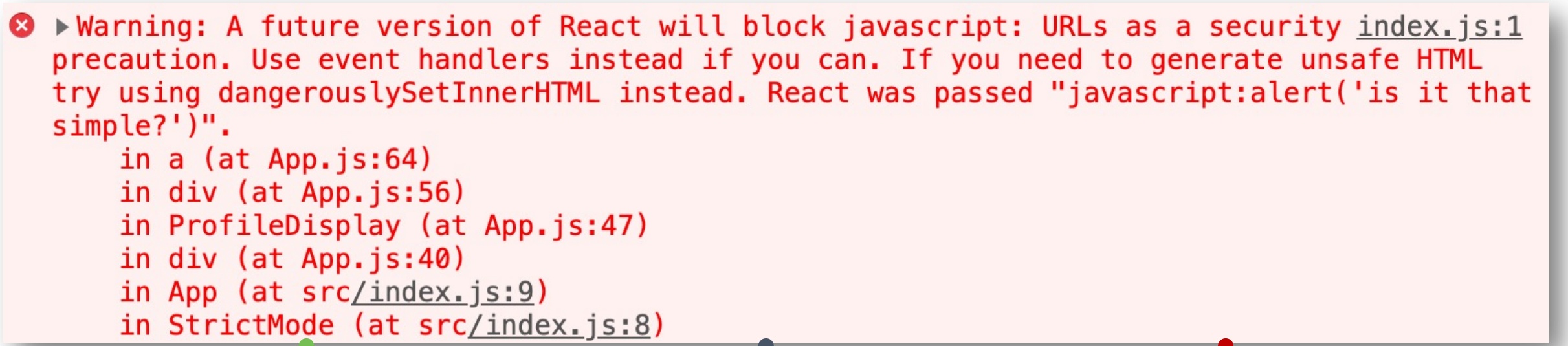
Creates a direct reference to a node in the DOM

Insecure direct DOM manipulation creates XSS vulnerabilities

Generating anchor tags with untrusted data

```
1  return (
2    <a href={url}>Open web page</a>
3  );
```





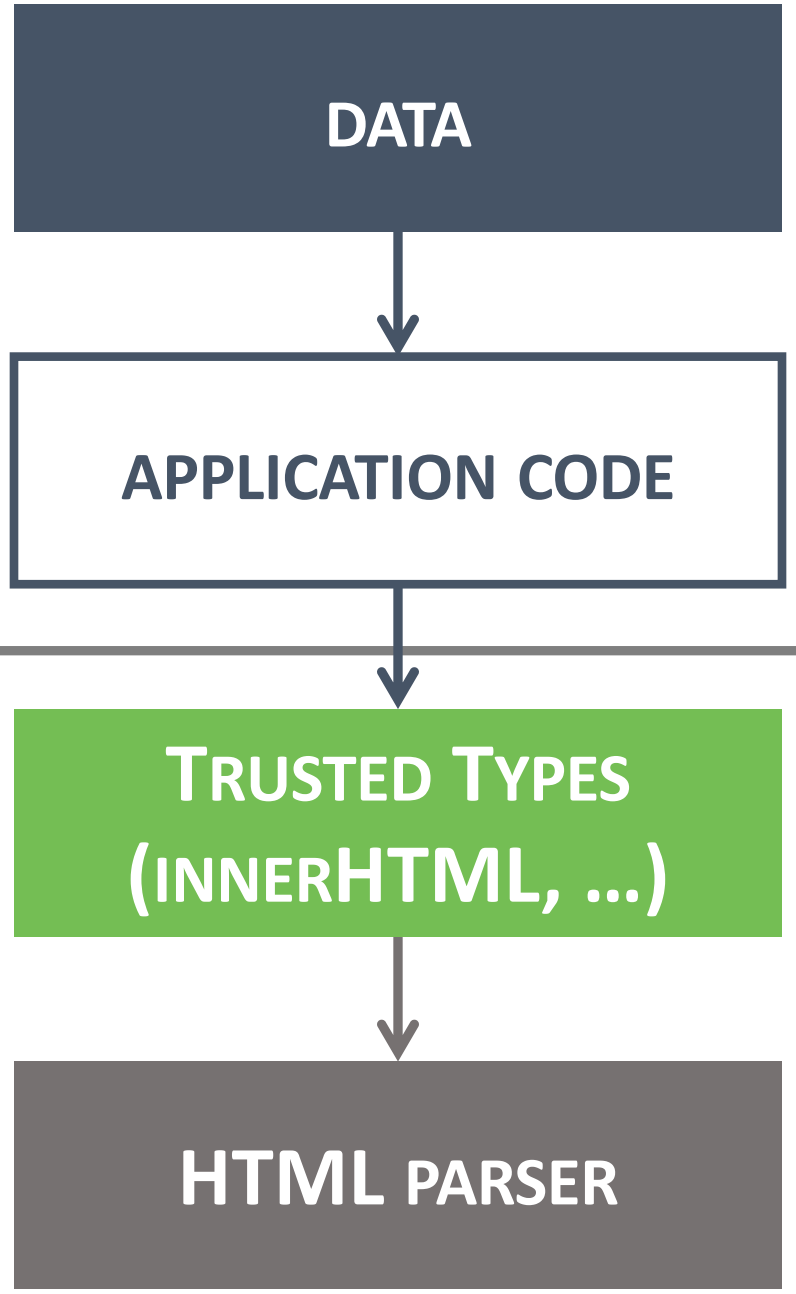
React detects this insecure behavior and warns about it in the developer tools

The warning has been there since March 12th, 2019, and is still here (React 17.0.2)

Even though the warning is an error, the dangerous URL is not blocked by React

Avoiding XSS in React is challenging

Making a single mistake when using a dangerous sink results in a dangerous XSS vulnerability



Enable trusted types by setting a CSP policy

```
1 Content-Security-Policy:
2   require-trusted-types-for 'script'
```

Application

Browser

```
▶Uncaught TypeError: Failed to set the 'innerHTML' property on 'Element': This document requires 'TrustedHTML' assignment.    index.js:1
    at HTMLIFrameElement.e.onload (index.js:1)
    at fe (index.js:1)
    at index.js:1
    at index.js:1
```

@PhilippeDeRyck

Trusted Types complement static analysis by providing runtime guarantees about the absence of uncontrolled data flows in client-side code. Our analysis of the vulnerabilities reported to Google VRP shows that Trusted Types could **effectively prevent at least 61% of DOM XSS-es** missed by our static analysis pipeline.

https://docs.google.com/document/d/1m91JZWKAGOR3jQoicMVE9Ydcq79gM2BetcRIBemrex8/view#

Enable trusted types by setting a CSP policy

```
1  Content-Security-Policy: require-trusted-types-for 'script'
```

Tells the browser to only allow trusted types in the DOM

Trusted Types does not affect the use of proper DOM APIs

```
1  let msg = document.createElement("span");
2  msg.setAttribute("class", "italic");
3  msg.textContent = e.data;
4  document.getElementById("msg").appendChild(msg);
```

When possible, always opt to write clean code instead of relying on the browser's HTML parser

ee greatest things you learn from traveling

the great things on earth traveling teaches us by example. Here are some of the mo
ous lessons I've learned over the years of traveling.

Leaving your comfort zone might lead you to such beautiful sceneries like this one.

ppreciation of diversity

tting used to an entirely different culture can be challenging. While it's also nice to learn abo
es online or from books, nothing comes close to experiencing cultural diversity in perso
n to appreciate each and every single one of the differences while you become mo
fluid.

Enable trusted types by setting a CSP policy

```
1  Content-Security-Policy: require-trusted-types-for 'script'
```

Tells the browser to only allow trusted types in the DOM

DOMPurify can generate Trusted Types when sanitizing

```
1  <p dangerouslySetInnerHTML={{__html:
2    DOMPurify.sanitize(review, {RETURN_TRUSTED_TYPE: true})}}></p>
```

DOMPurify can return a trusted type, which is allowed to be assigned to *innerHTML*

DATA

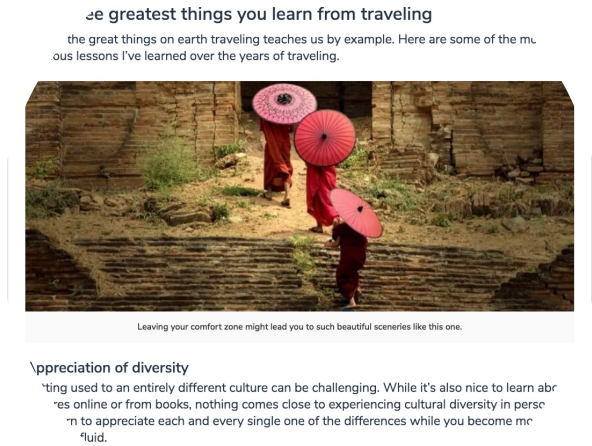
escaping / sanitization

Application

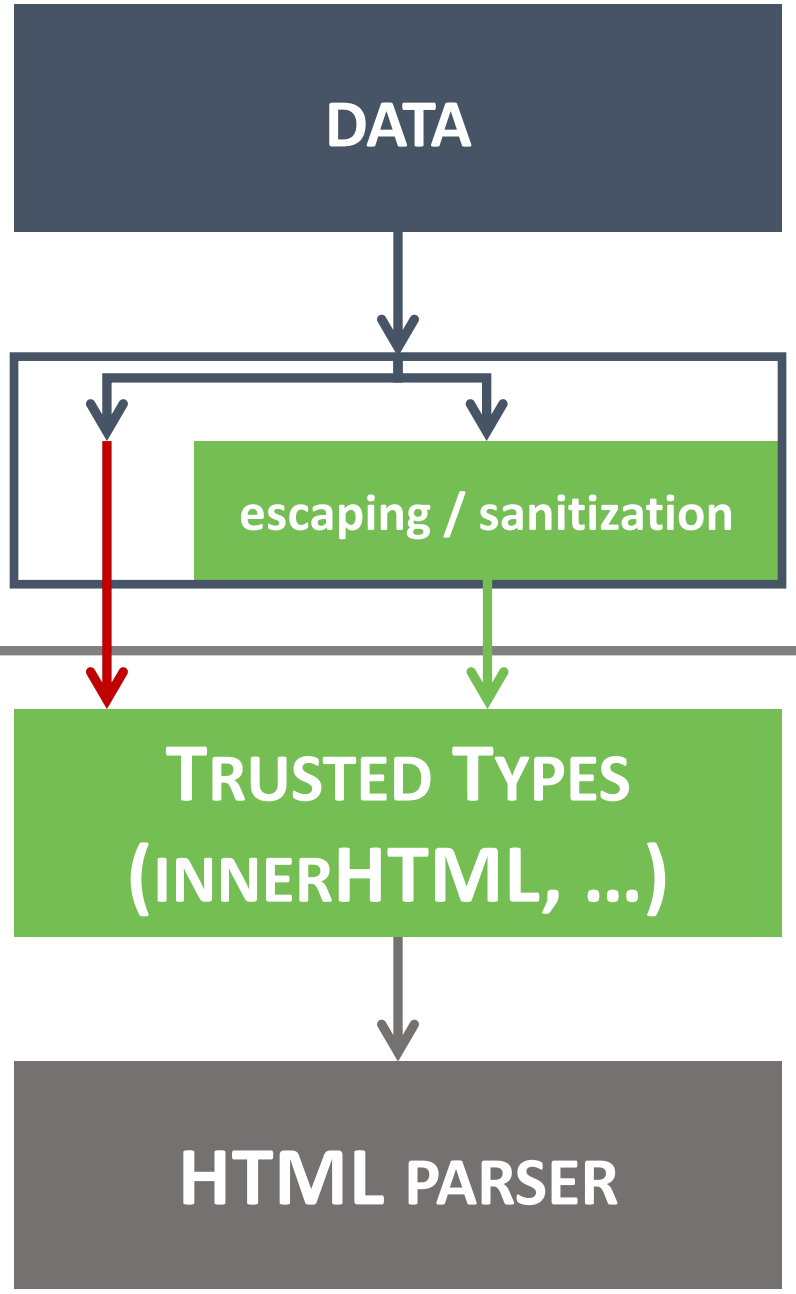
Browser

TRUSTED TYPES (INNERHTML, ...)

HTML PARSER



@PhilippeDeRyck




```
▶Uncaught TypeError: Failed to set the  index.js:1
'innerHTML' property on 'Element': This document
requires 'TrustedHTML' assignment.
    at HTMLIFrameElement.e.onload (index.js:1)
    at fe (index.js:1)
    at index.js:1
    at index.js:1
```

Trusted Types **improve the security** of applications. To date, we have observed zero DOM XSS in Google applications migrated to Trusted Types. Around ¾ of code locations incompatible with Trusted Types can be addressed by mechanical changes to rewrite the code to be inherently safe and not use a DOM XSS sink. Future web APIs (e.g. the HTML Sanitizer API) may tip the scale even further. Some web applications may be

https://docs.google.com/document/d/1m91JZWKAGOR3jQoicMVE9Ydcq79gM2BetcRIBemrex8/view#

TT forces you to transform text to a Trusted Type, it does not automatically apply security

Define a TT policy that transforms unsafe input into safe output

```
<script>
  const myTTpolicy = trustedTypes.createPolicy('myTTpolicy', {
    //Transform the 'string' variable into safe output
    createHTML: (string) => DOMPurify.sanitize(string)
  });
</script>
```

Manipulate the incoming data to make it safe

Enable trusted types by setting a CSP policy

```
Content-Security-Policy: require-trusted-types-for 'script'; trusted-types myTTpolicy
```

Allow the use of the ***myTTpolicy*** for turning untrusted data into safe data

Use the application's TT policy to make untrusted data safe

```
<p dangerouslySetInnerHTML={{__html: myTTpolicy.createHTML(review)}}></p>
```

Define a TT policy that transforms unsafe input into safe output

```
1  <script>
2    const badIdea = trust          atePolicy('realMen       anitize', {
3      //Transform the 'stri          e into a Trust
4      createHTML: (string) =>
5    });
6  </script>
```

Simply returning the data makes Trusted Types useless

Enable trusted types by setting a CSP policy

```
1  Content-Security-Policy:
2    require-trusted-types-for 's          ted-          MenDoNotSanitize
```

Allow th
realMenDoNotSaniti **turning**
untrusted data into te type

Use a custom Trusted Types policy in the application

```
1  <p dangerouslySetInnerHTML={{__html: badIdea.createHTML(review)}}></p>
```

Trusted Types avoids unsafe DOM assignments

Enabling Trusted Types modifies default browser behavior, refusing the insecure usage of dangerous sinks in the DOM

Trusted Types for DOM manipulation

- UNOFF

Usage % of all users ?

Global 71.39%

An API that forces developers to be very explicit about their use of powerful DOM-injection APIs. Can greatly improve security against XSS attacks.

Current aligned | Usage relative | Date relative | Filtered | All

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *	Opera Mobile *
	12-81		4-81		10-68				
6-10	83-95	2-93	83-95	3.1-15	69-80	3.2-14.8		2.1-4.4.4	12-12.1
11	96	94	96	15.1	81	15	all	96	64
		95-96	97-99	TP					

Source: caniuse.com

Photo by Dušan veverkolog on Unsplash

Photo by Dušan veverkolog on Unsplash

Enable trusted types by setting a CSP policy

```
Content-Security-Policy: require-trusted-types-for 'script'
```

With trusted types enabled, the browser refuses to assign text to innerHTML

```
<p dangerouslySetInnerHTML={{__html: review}}></p>
```

Fixing the application for Chrome typically results in applying proper protections

```
<p dangerouslySetInnerHTML={{__html:
  DOMPurify.sanitize(review, {RETURN_TRUSTED_TYPE: true})}}></p>
```

Enable trusted types by setting a CSP policy

```
1  Content-Security-Policy: require-trusted-types-for 'script'
```

Thanks to trusted types, the application relies on DOMPurify to sanitize the data

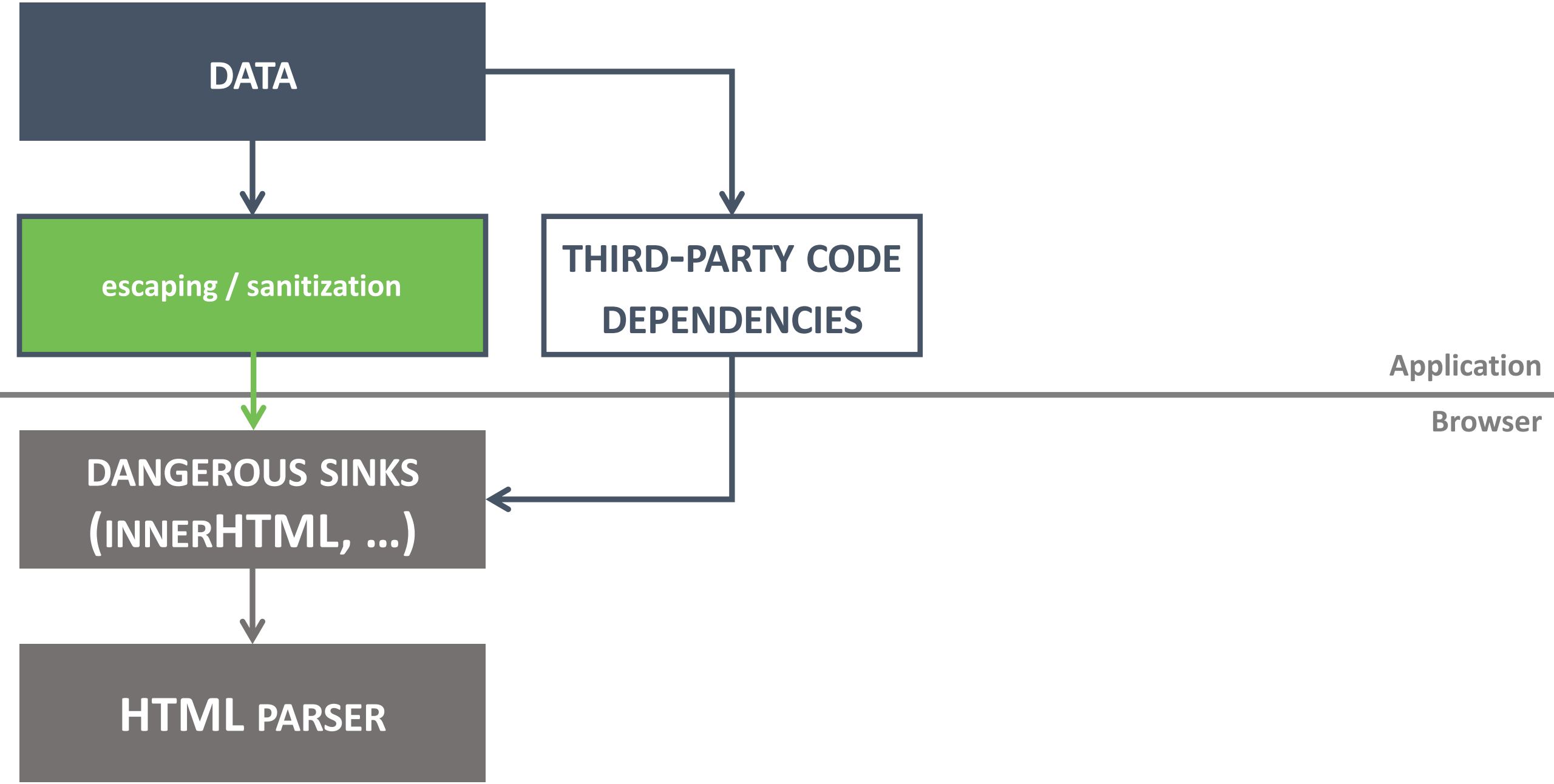
```
1  <p dangerouslySetInnerHTML={{__html:
2    DOMPurify.sanitize(review, {RETURN_TRUSTED_TYPE: true})}}></p>
```

Enabling Trusted Types automatically results in better coding practices, even when only used in development

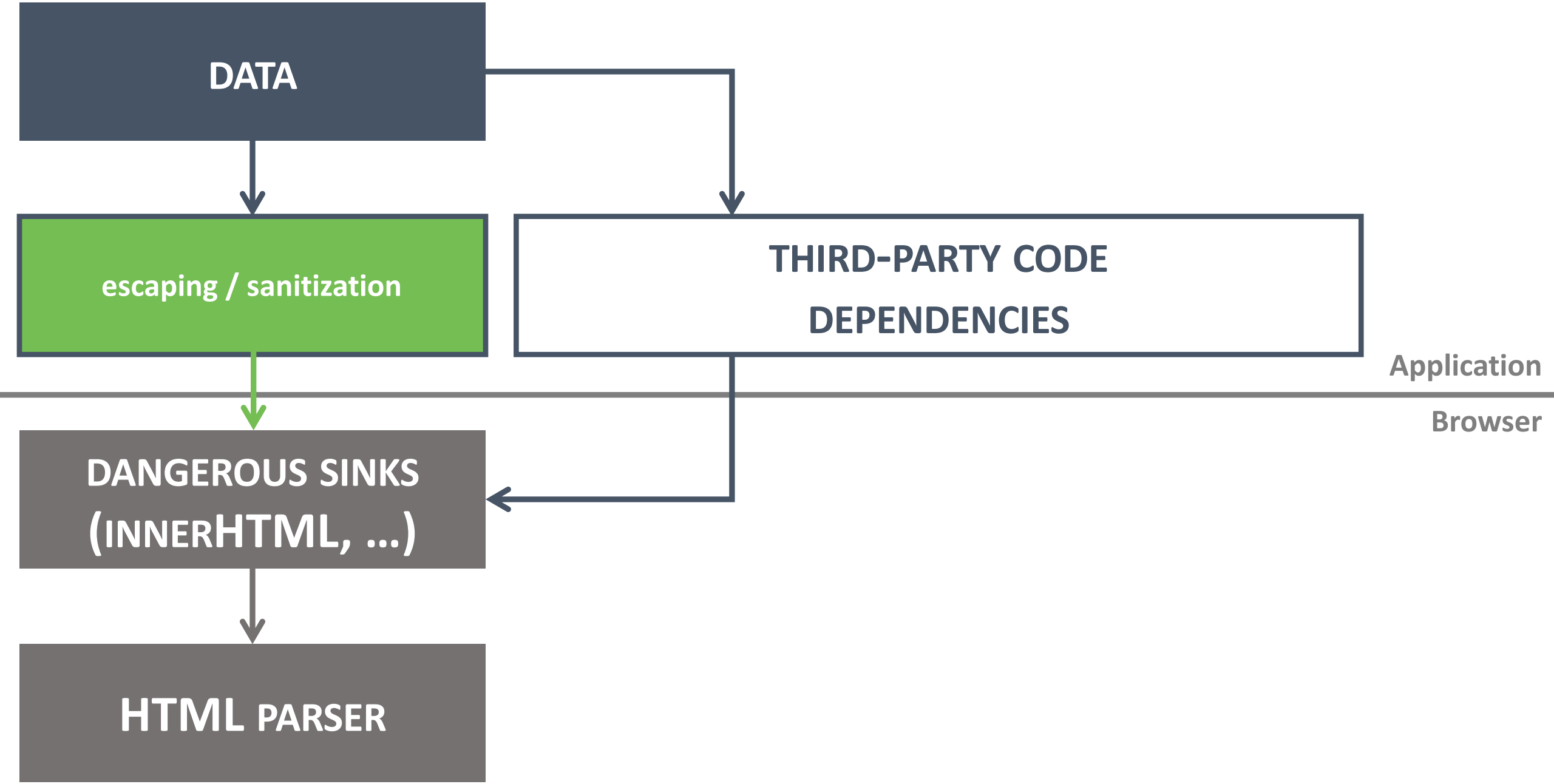
Trusted Types polyfills are available for non-supporting browsers

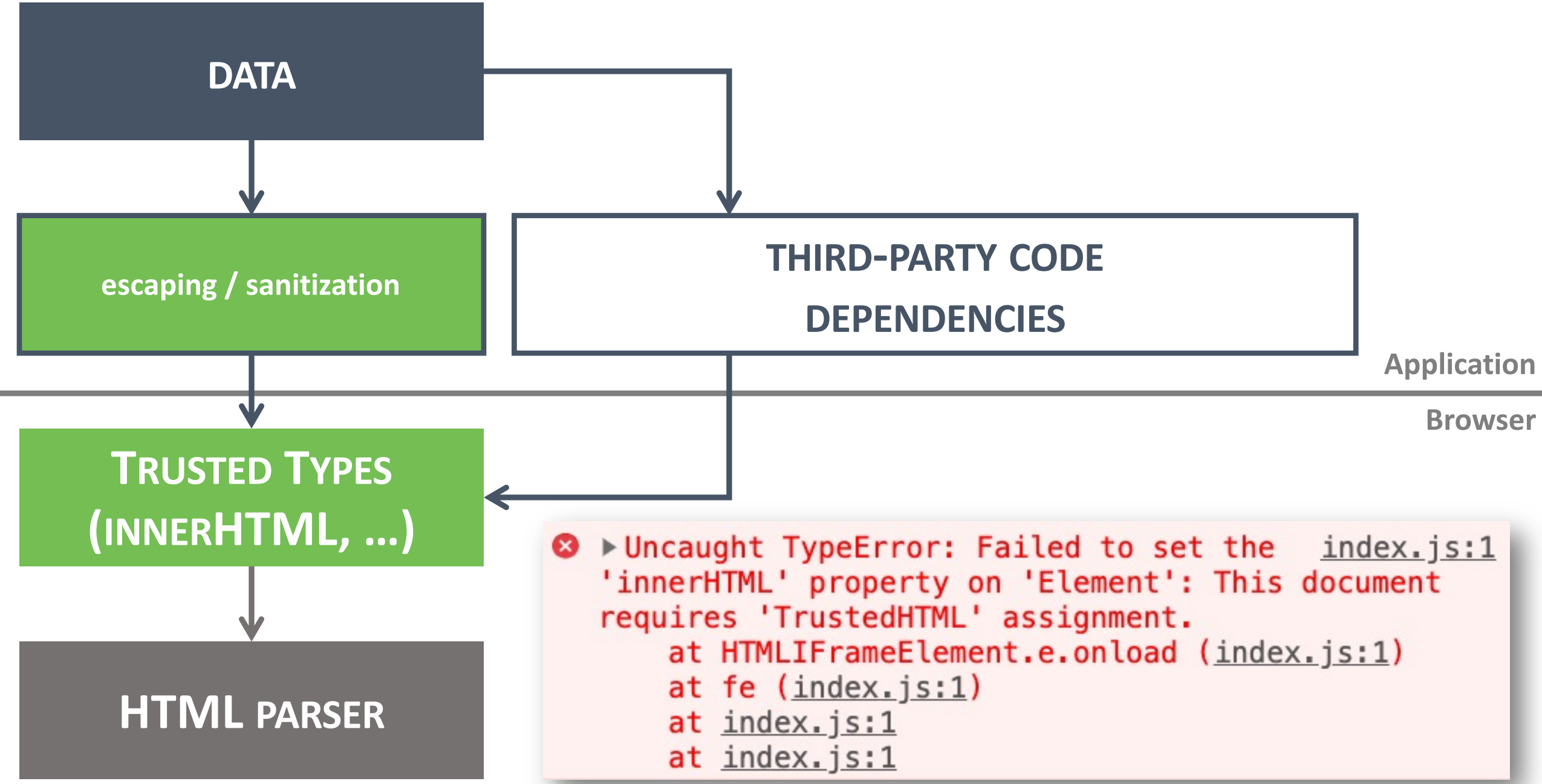
Trusted Types improves code security

Having Trusted Types point out unsafe assignments to the DOM helps fixing these issues in the application's code, benefiting all users



@PhilippeDeRyck


DATA
escaping / sanitization
THIRD-PARTY CODE
DEPENDENCIES
Application
Browser
DANGEROUS SINKS
(INNERHTML, ...)
HTML PARSER


DATA
escaping / sanitization
THIRD-PARTY CODE
DEPENDENCIES
Application
Browser
TRUSTED TYPES
(INNERHTML, ...)
HTML PARSER
Uncaught TypeError: Failed to set the 'innerHTML' property on 'Element': This document requires 'TrustedHTML' assignment.
at HTMLIFrameElement.e.onload (index.js:1)
at fe (index.js:1)
at index.js:1
at index.js:1
index.js:1

Enable trusted types by setting a CSP policy

```
Content-Security-Policy: require-trusted-types-for 'script'
```

Specify a default TT policy that is applied on all text assigned to HTML sinks

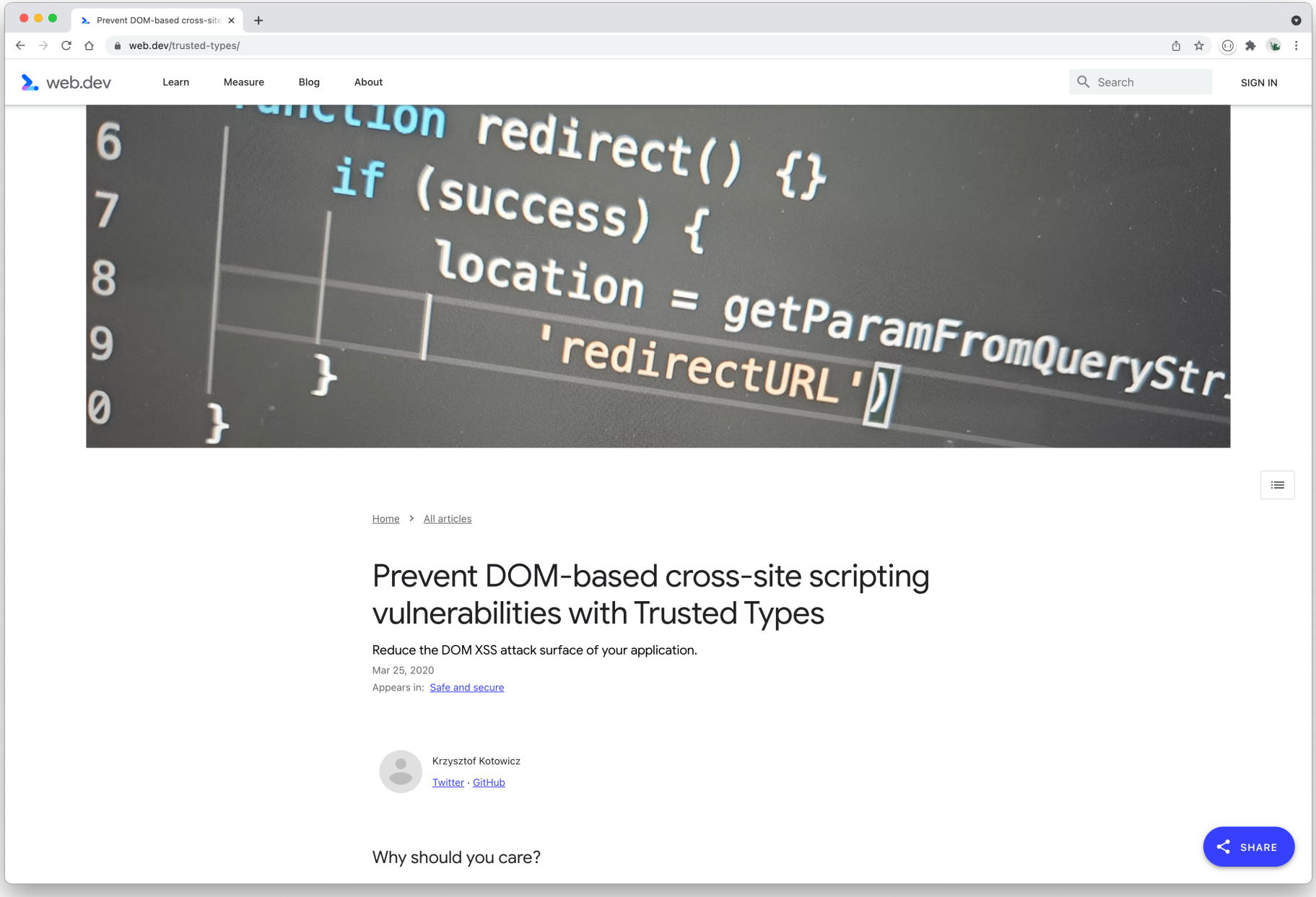
```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/dompurify/2.2.4/purify.min.js"></script>
<script>
  //Define a default policy
  trustedTypes.createPolicy('default', {
    createHTML: (string, sink) =>
      DOMPurify.sanitize(string, {RETURN_TRUSTED_TYPE: true})
  });
</script>
```

Defining a default policy automatically applies the *createHTML* function on string-based assignments to *innerHTML*, which fixes the application

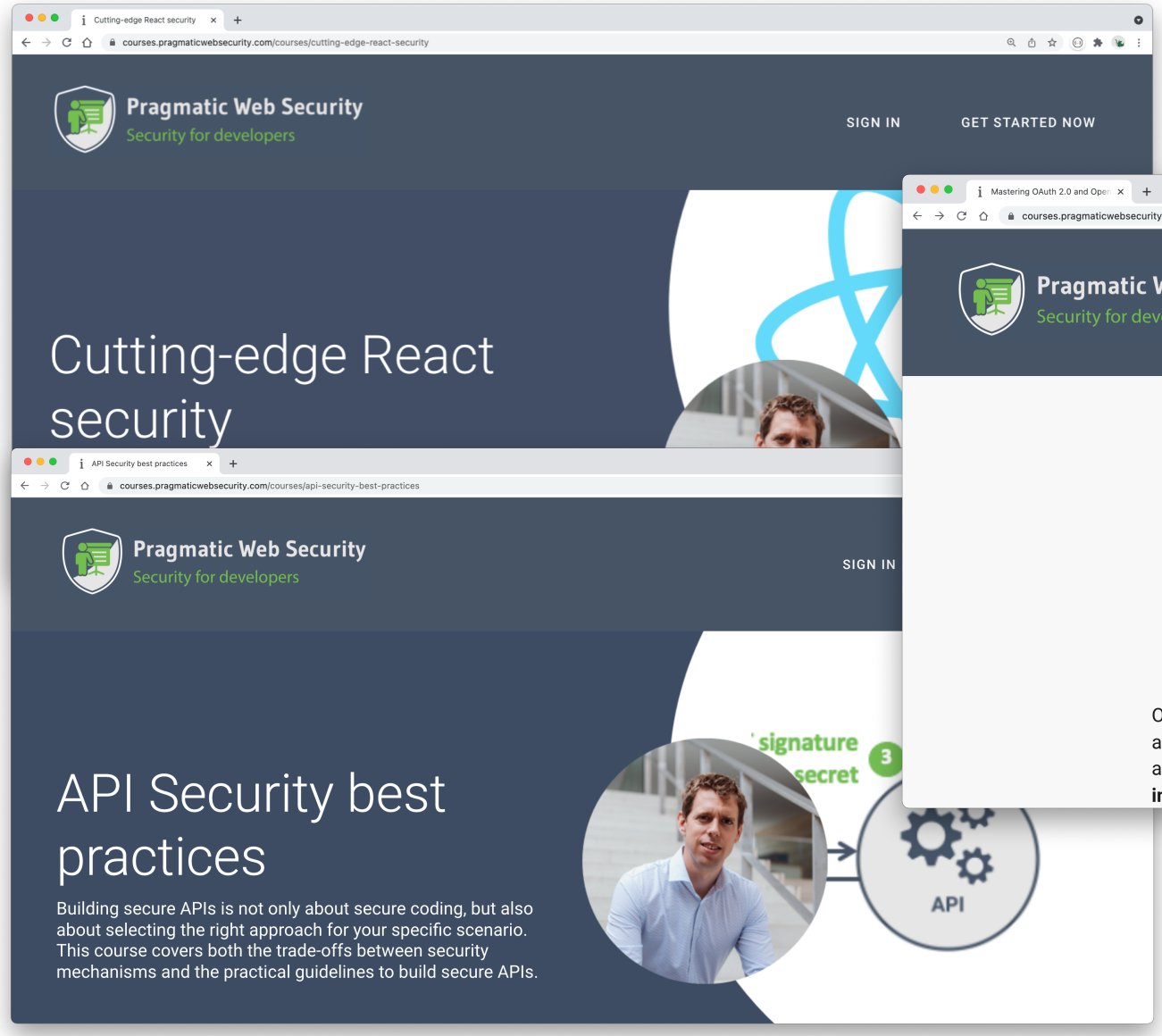
A default Trusted Types policy requires browser support or the loading of the polyfill

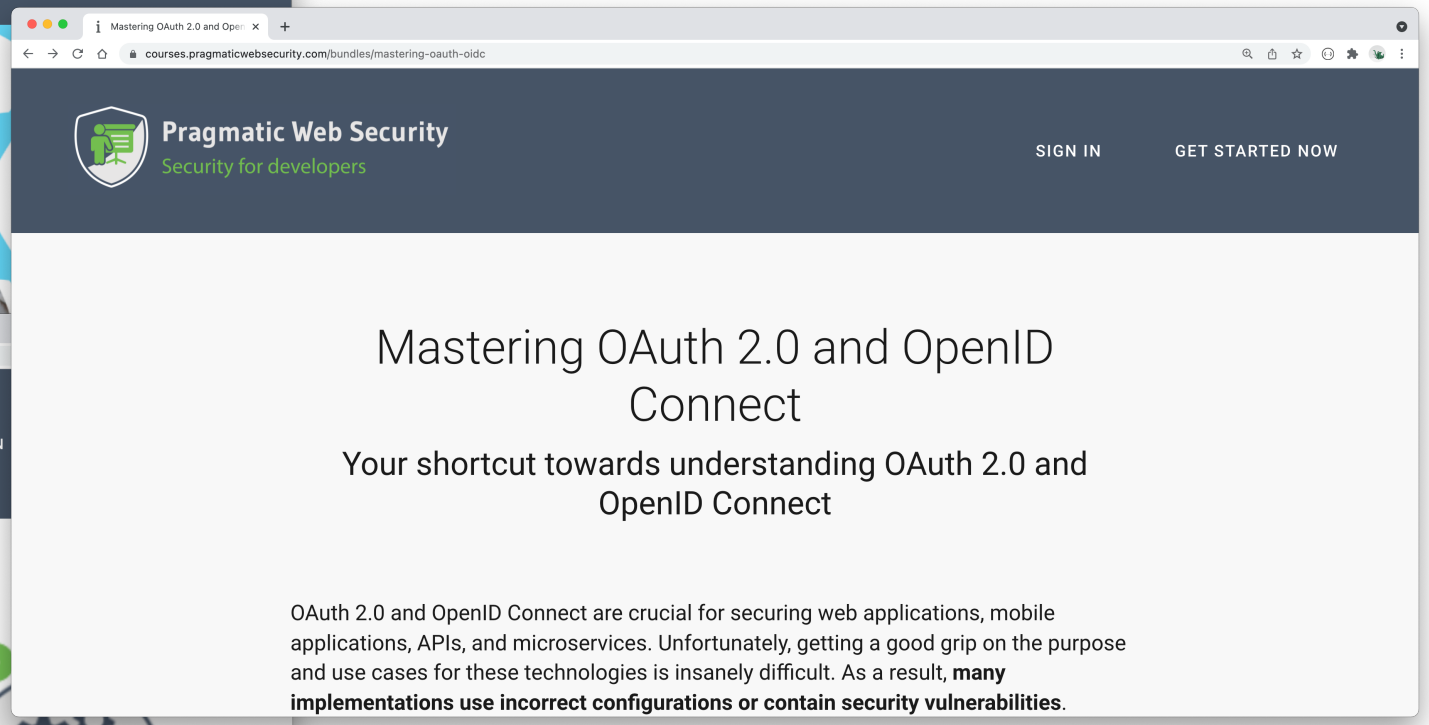
Trusted Types is a browser-level measure

Trusted Types prevents that third-party code or dependencies from using dangerous sinks, and a default policy can automatically enable protection


Prevent DOM-based cross-site
web.dev/trusted-types/
web.dev
Learn
Measure
Blog
About
Search
SIGN IN
function redirect() {}
if (success) {
location = getParamFromQueryStr
'redirectURL')
}
}
Home
All articles
Prevent DOM-based cross-site scripting vulnerabilities with Trusted Types
Reduce the DOM XSS attack surface of your application.
Mar 25, 2020
Appears in: Safe and secure
Krzysztof Kotowicz
Twitter · GitHub
Why should you care?
SHARE

Want more in-depth security content?





HTTPS://COURSES.PRAGMATICWEBSECURITY.COM

Thank you!

Connect on social media to stay in touch on security

@PhilippeDeRyck

/in/PhilippeDeRyck