



SECURING FRONTENDS WITH TRUSTED TYPES

DR. PHILIPPE DE RYCK

<https://PragmaticWebSecurity.com>

Philippe De Ryck

<h1>

Welcome Philippe De Ryck

</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

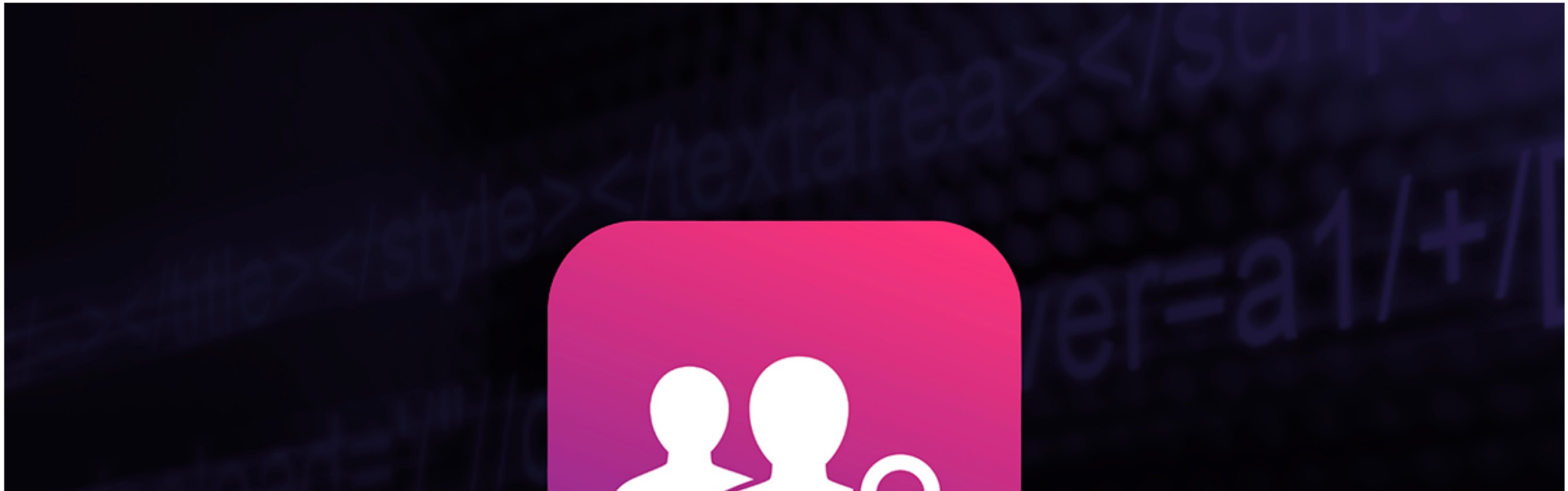
XSS

Vulnerabilities

Mobile



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



@PhilippeDeRyck

<https://portswigger.net/daily-swig/multiple-xss-vulnerabilities-in-child-monitoring-app-canopy-could-risk-location-leak>

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



@PhilippeDeRyck

<https://portswigger.net/daily-swig/facebook-pays-out-25k-bug-bounty-for-chained-dom-based-xss>



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

I am *Dr. Philippe De Ryck*



Founder of Pragmatic Web Security



Google Developer Expert



Auth0 Ambassador



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

An Angular template to put data into the page

```
1 <h1> Welcome {{ name }} </h1>
```

Angular/React/Vue/Ember
apply automatic escaping to
data embedded in templates

The data seen by the browser

```
1 Philippe De Ryck
```

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



Some of the greatest things you learn from traveling

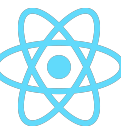
One of the great things on earth traveling teaches us by example. Here are some of the most precious lessons I've learned over the years of traveling.



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

Appreciation of diversity

Getting used to an entirely different culture can be challenging. While it's also nice to learn about cultures online or from books, nothing comes close to experiencing cultural diversity in person. It helps you learn to appreciate each and every single one of the differences while you become more open and fluid.



DATA



DANGEROUSLYSETINNERHTML



DANGEROUS SINKS
(INNERHTML, ...)



HTML PARSER



Application

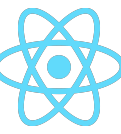
Browser



@PhilippeDeRyck

Appreciation of diversity

Using used to an entirely different culture can be challenging. While it's also nice to learn about cultures online or from books, nothing comes close to experiencing cultural diversity in person. It's important to appreciate each and every single one of the differences while you become more fluid.



Philippe De Ryck



DANGEROUSLYSETINNERHTML



 LOL, dangerous

innerHTML



HTML parser



LOAD IMAGE AND
TRIGGER ERROR



JS ENGINE



MALICIOUS CODE
EXECUTION

Application
Browser



DATA



escaping / sanitization



**DANGEROUS SINKS
(INNERHTML, ...)**

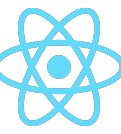


HTML PARSER

Application

Browser





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>);
```

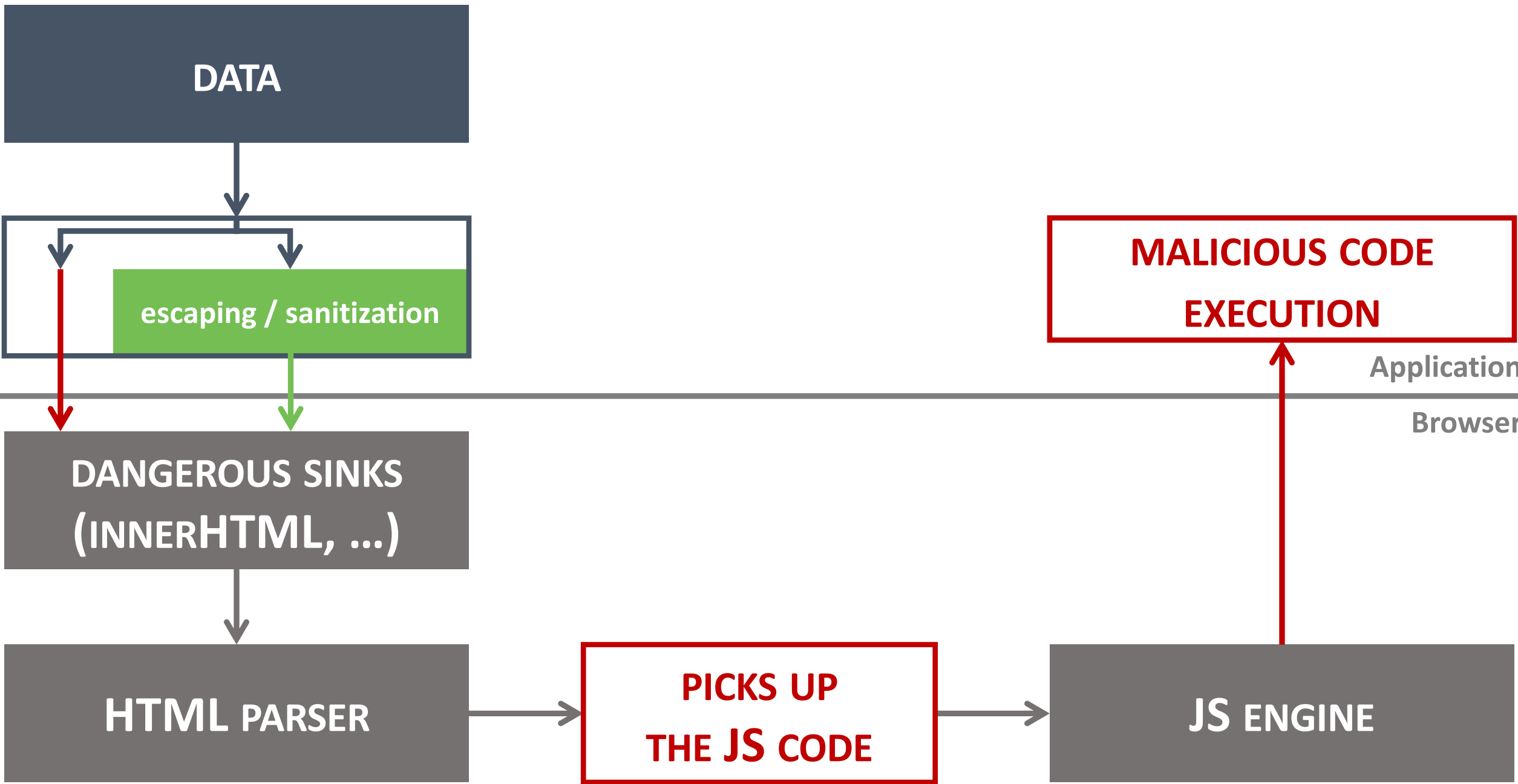
**DOMPurify turns untrusted
HTML in safe HTML, making it
safe to include in the page**



An Angular template to render user-provided HTML

```
1 <div>
2   <h3>{{ review.title }}</h3>
3   <p [innerHTML]="review.content"></p>
4 </div>
```

[innerHTML] does not directly expose **innerHTML** property, but sanitizes the data first





Signal Messenger

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

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

```
115         );
116     }
```

```
117
```

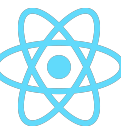
```
@@
```

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

```
119         );
120     }
```

```
121
```





Using React 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**



Angular code to obtain a native DOM element

```
1 @ViewChild("myDiv") div : ElementRef;
```

With Angular out of the loop, bad things are bound to happen

```
1 this.div.nativeElement.innerHTML = this.inputValue;
```

With *ElementRef*, you can access native DOM elements, where Angular cannot apply automatic protection against XSS

XSS IN FRONTENDS IS STILL A REALITY



Frontend frameworks make it harder to cause XSS vulnerabilities, but a single mistake can still compromise the security of the application



DATA



APPLICATION CODE



TRUSTED TYPES
(innerHTML, ...)



HTML PARSER

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 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 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.



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



Some of the greatest things you learn from traveling

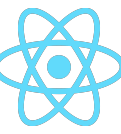
One of the great things on earth traveling teaches us by example. Here are some of the most precious lessons I've learned over the years of traveling.



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

Appreciation of diversity

Getting used to an entirely different culture can be challenging. While it's also nice to learn about cultures online or from books, nothing comes close to experiencing cultural diversity in person. It helps you learn to appreciate each and every single one of the differences while you become more open and 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*



Enforcing Trusted Types

We recommend the use of [Trusted Types](#) as a way to help secure your applications from cross-site scripting attacks. Trusted Types is a [web platform](#) feature that can help you prevent cross-site scripting attacks by enforcing safer coding practices. Trusted Types can also help simplify the auditing of application code.

Trusted Types might not yet be available in all browsers your application targets. In the case your Trusted-Types-enabled application runs in a browser that doesn't support Trusted Types, the functionality of the application will be preserved, and your application will be guarded against XSS via Angular's DomSanitizer. See caniuse.com/trusted-types for the current browser support.

To enforce Trusted Types for your application, you must configure your application's web server to emit HTTP headers with one of the following Angular policies:

- `angular` - This policy is used in security-reviewed code that is internal to Angular, and is required for Angular to function when Trusted Types are enforced. Any inline template values or content sanitized by Angular is treated as safe by this policy.
- `angular#unsafe-bypass` - This policy is used for applications that use any of the methods in Angular's [DomSanitizer](#) that bypass security, such as `bypassSecurityTrustHtml`. Any application that uses these methods must enable this policy.
- `angular#unsafe-jit` - This policy is used by the [JIT compiler](#). You must enable this policy if your application interacts directly with the JIT compiler or is running in JIT mode using the [platform browser dynamic](#).





Enable trusted types by setting a CSP policy

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

Tells the browser to enable
Trusted Types and allow values
returned by the Angular policy

Angular's sanitizer automatically generates Trusted Types

```
1 <p [innerHTML]="review.content"></p>
```

Angular automatically returns a trusted type,
making it compatible with TT out of the box





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

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

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

Usage

% of all users



Global

70.18%

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	83-97	2-96	83-97	3.1-15.1	69-82	3.2-15.1		2.1-4.4.4	12-12.1
11	98	97	98	15.3	83	15.3	all	98	64
		98-99	99-101	15.4-TP		15.4			

Enable trusted types by setting a CSP policy

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



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

1 `this.div.nativeElement.innerHTML = this.inputValue;`



Fixing the application for Chrome typically results in applying proper protections

1 `<div [innerHTML]="inputValue"></div>`



Enable trusted types by setting a CSP policy

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



Thanks to trusted types, the application follows security best practices

```
1 <div [innerHTML]="inputValue"></div>
```

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





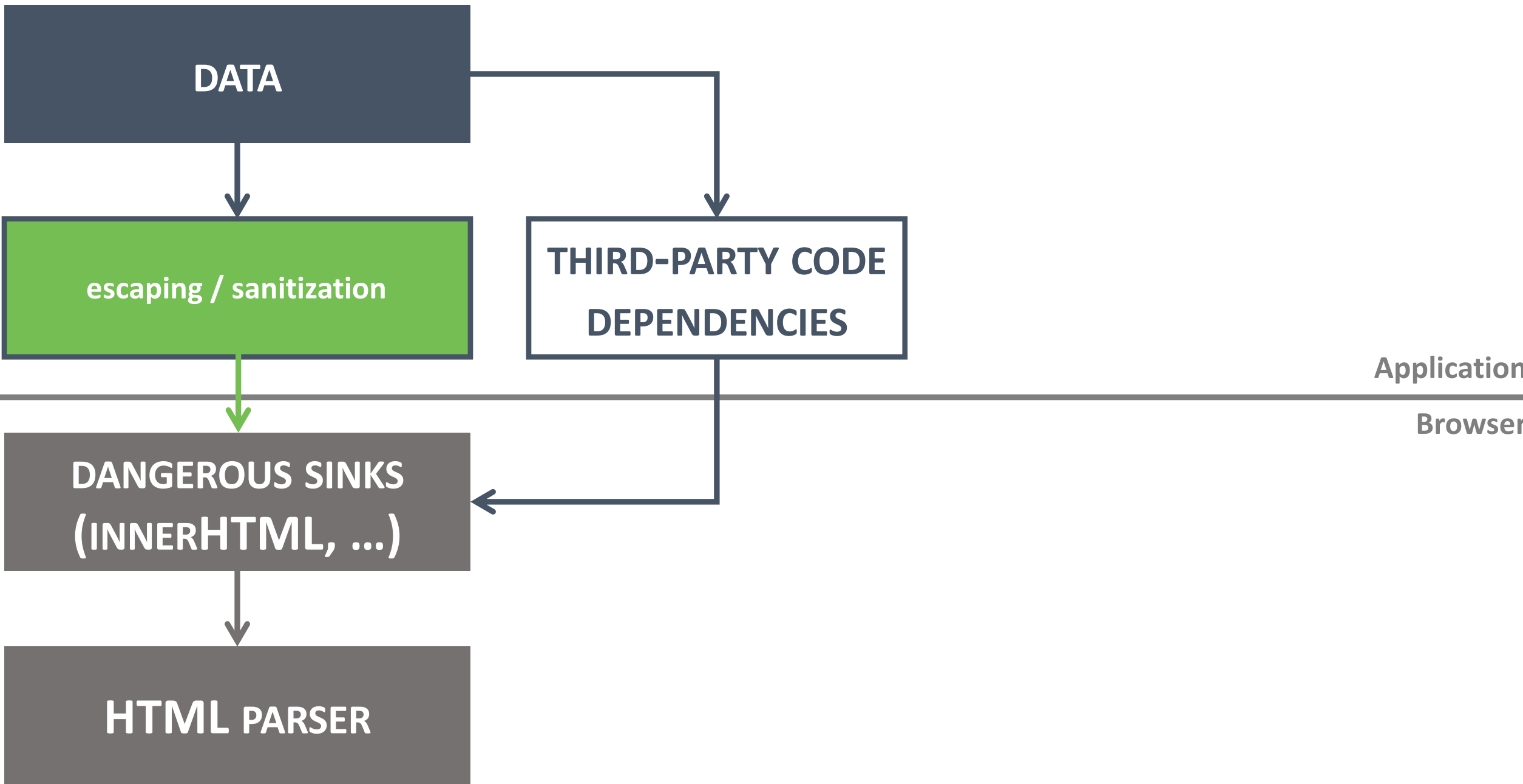
**Trusted Types polyfills are available
for non-supporting browsers**

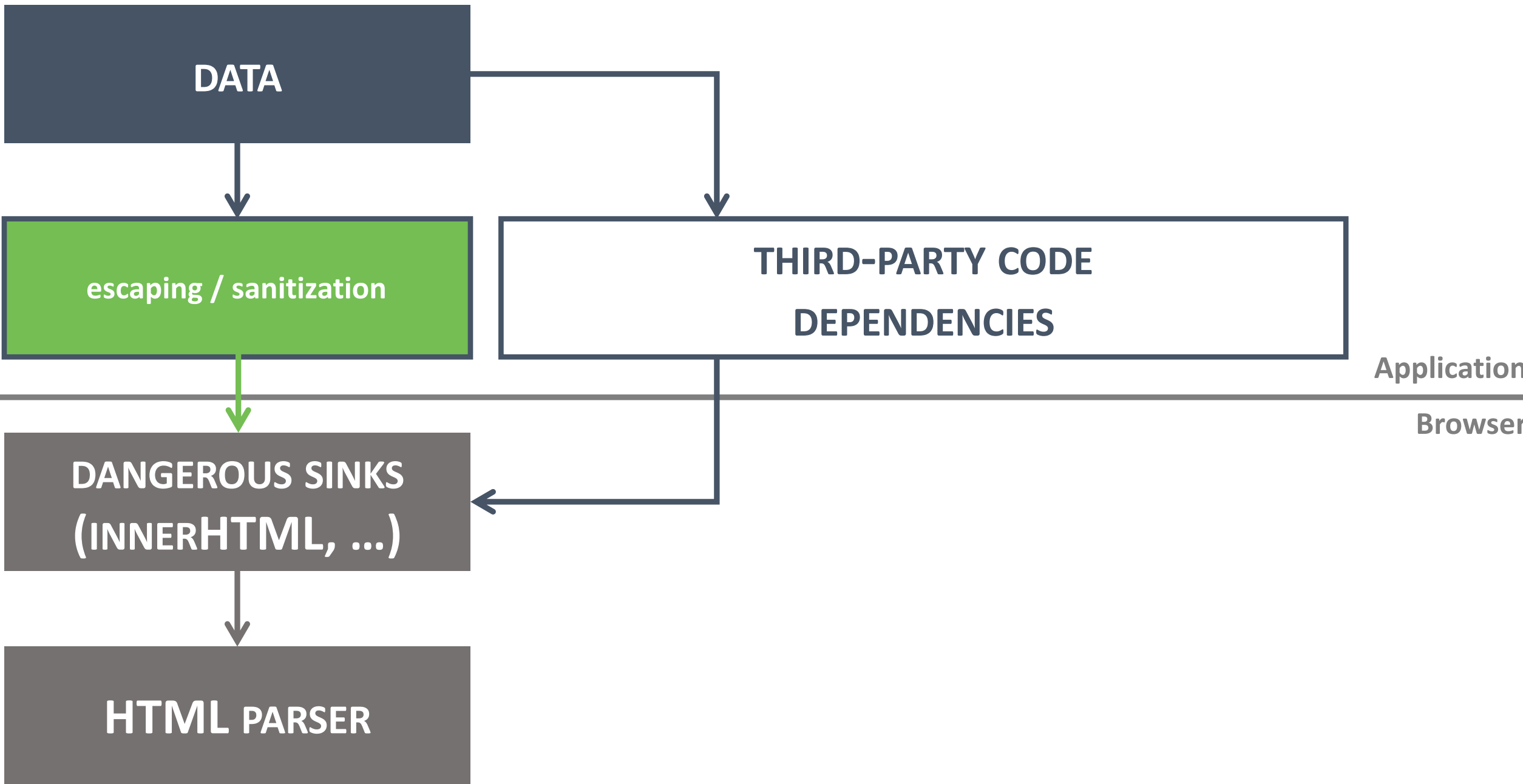
TRUSTED TYPES IMPROVES CODE SECURITY

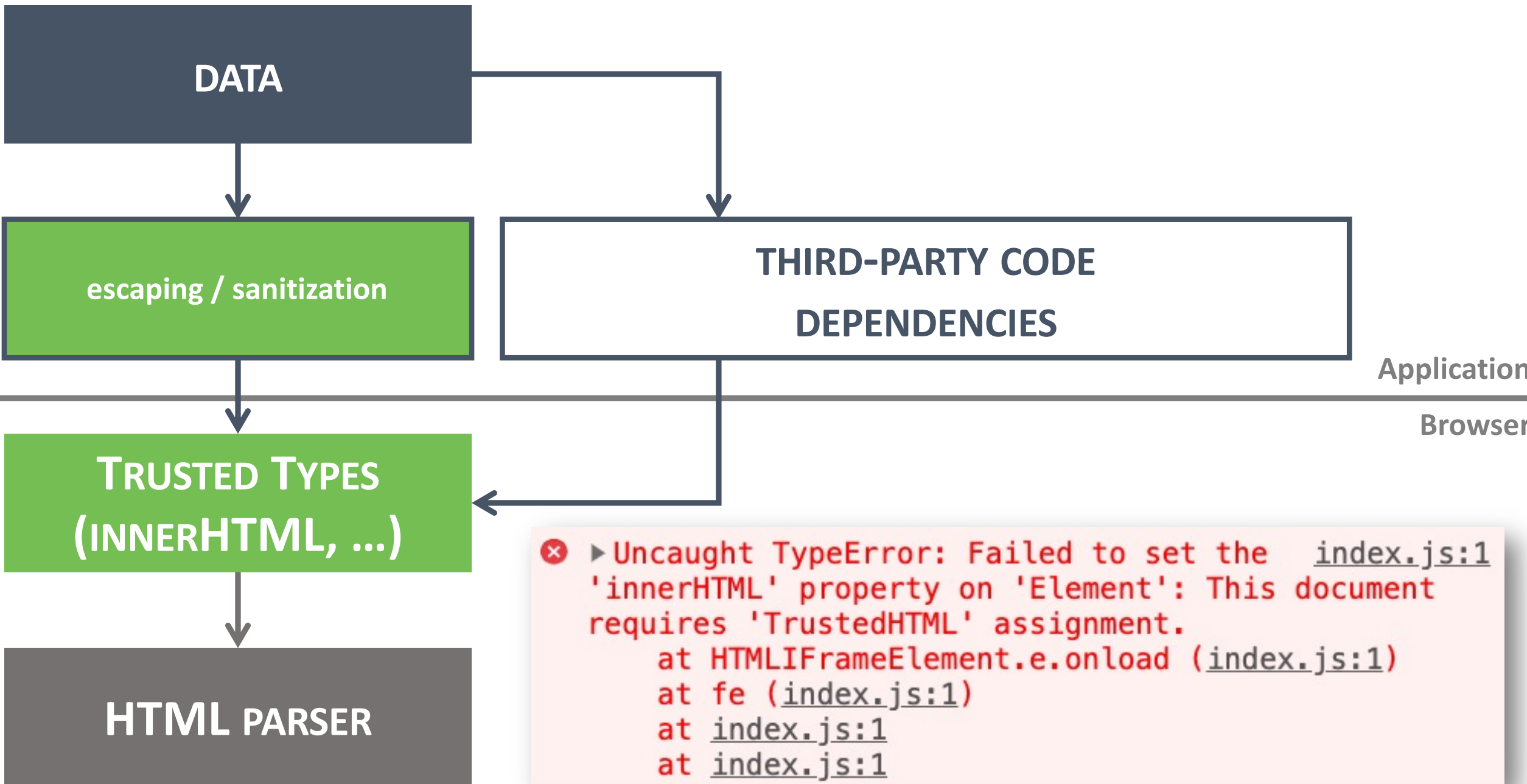


Having Trusted Types point out unsafe assignments to the DOM helps fixing these issues in the application's code, benefiting all users









Enable trusted types by setting a CSP policy

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

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

```
1 <script>
2 src="https://cdnjs.cloudflare.com/ajax/libs/dompurify/2.2.4/purify.min.js"></script>
3 <script>
4   //Define a default policy
5   trustedTypes.createPolicy('default', {
6     createHTML: (string, sink) =>
7       DOMPurify.sanitize(string)
8   });
9 </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 X

web.dev/trusted-types/

web.dev

LearnMeasureBlogAbout

Search

SIGN IN

```
6 function redirect() {}
7   if (success) {
8     location = getParamFromQueryString(
9       'redirectURL')
10  }
```


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](#)



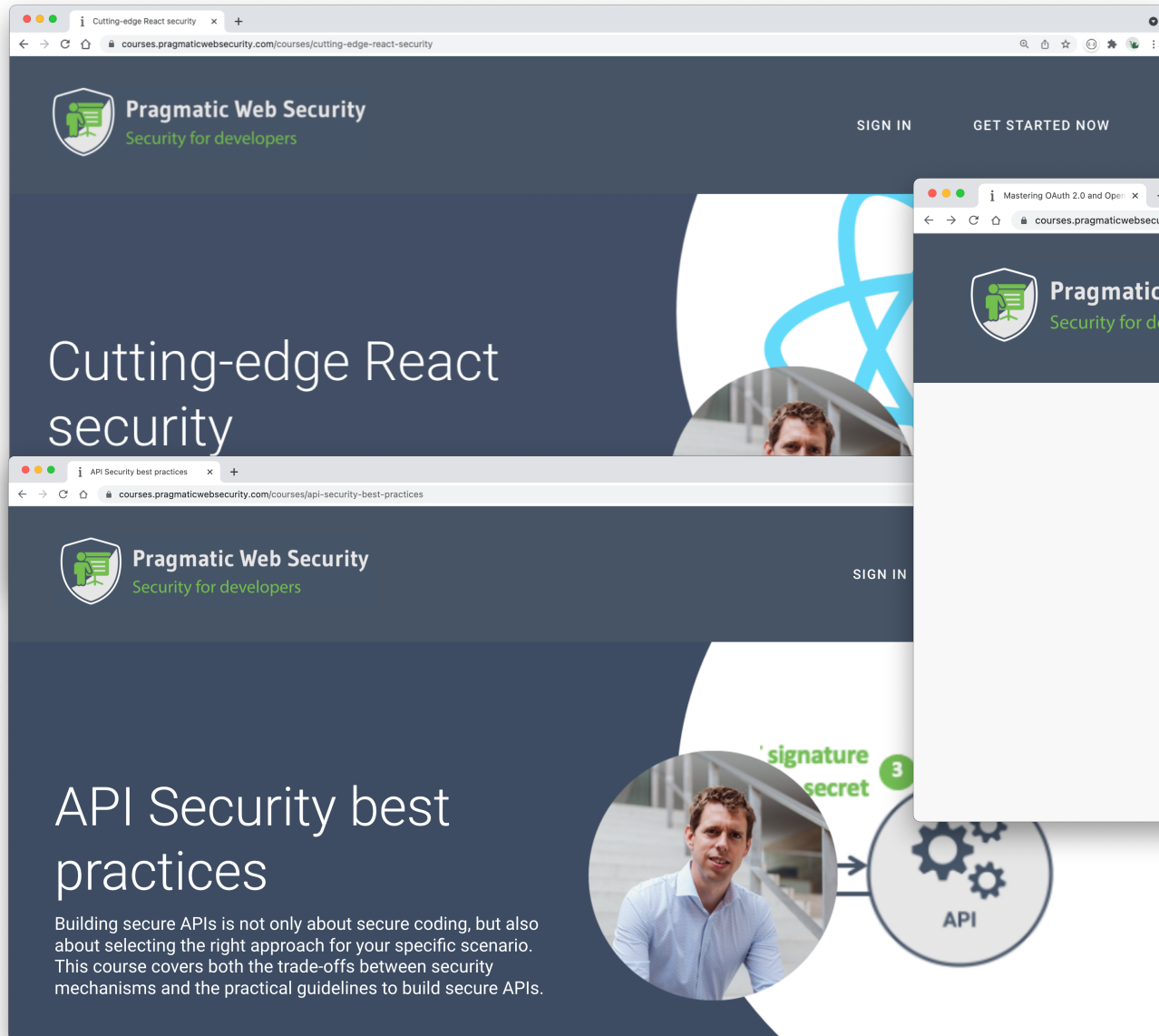
Krzysztof Kotowicz

[Twitter](#) · [GitHub](#)

Why should you care?

SHARE

Want more in-depth security content?



Mastering OAuth 2.0 and OpenID Connect

Your shortcut towards understanding OAuth 2.0 and OpenID Connect

OAuth 2.0 and OpenID Connect are crucial for securing web applications, mobile applications, APIs, and microservices. Unfortunately, getting a good grip on the purpose and use cases for these technologies is insanely difficult. As a result, **many implementations use incorrect configurations or contain security vulnerabilities.**

[HTTPS://COURSES.PRAGMATICWEBSECURITY.COM](https://courses.pragmaticwebsecurity.com)



Thank you!

Always happy to connect
on social media



@PhilippeDeRyck



/in/PhilippeDeRyck

