Better security through code hygiene

Dr. Philippe De Ryck

https://Pragmatic Web Security.com

A template to render HTML in the application

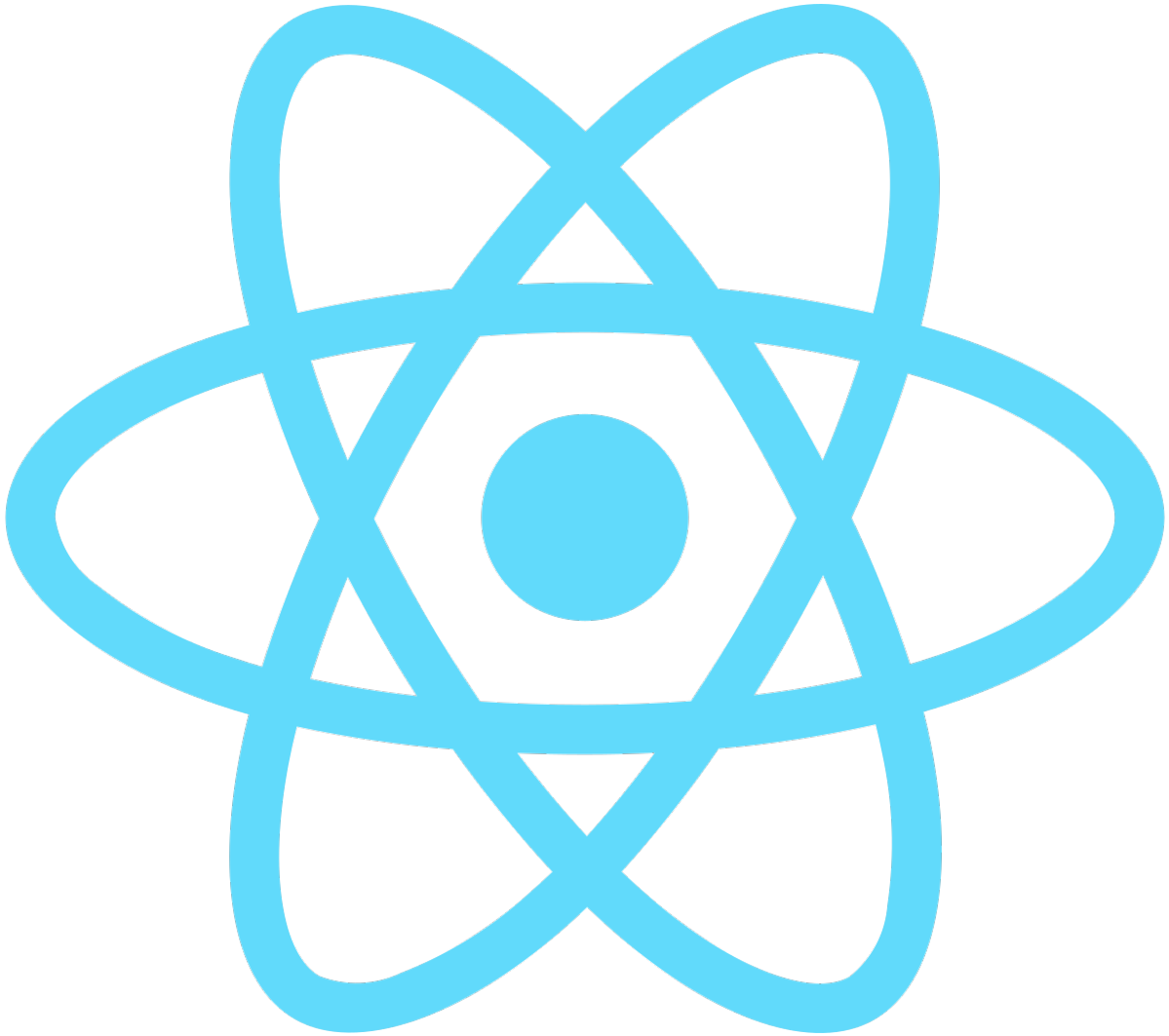
```
1  <h3>${title}</h3>
2  <p>${review}</p>
```

A review submitted by a malicious user

```
1  This restaurant is highly recommended. The food
2  is exquisite and the service is impeccable.
3  <script>alert('Go there, now!');</script>
```

The HTML page sent to the browser

```
1  <h3>Absolutely awesome</h3>
2  <p>This restaurant is highly recommended. The
3  food is exquisite and the service is impeccable.
4  <script>alert('Go there, now!');</script></p>
```



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

A JSX template to combine data with HTML

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

By default, React escapes values embedded in JSX before rendering them

A review submitted by a malicious user

```
1  This restaurant is <b>highly recommended</b>. The
2  food is exquisite and the service is impeccable. <a
3  href="https://pics.example.com">Check out my story
4  here!</a><img src="none.png" onerror="alert('Go
5  there, now!')">
```

https://restograde.com

Absolutely awesome

This restaurant is highly recommended. The food is exquisite and the service is impeccable. Check out my story here!<img src="none.png" onerror="alert('Go there, now!')">

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.

A JSX template to render user-provided HTML

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

dangerouslySetInnerHTML **exposes the *innerHTML* property**

A review submitted by a malicious user

```
1  This restaurant is <b>highly recommended</b>. The
2  food is exquisite and the service is impeccable. <a
3  href="https://pics.example.com">Check out my story
4  here!</a><img src="none.png" onerror="alert('Go
5  there, now!')">
```

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

A JSX template to render user-provided HTML

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

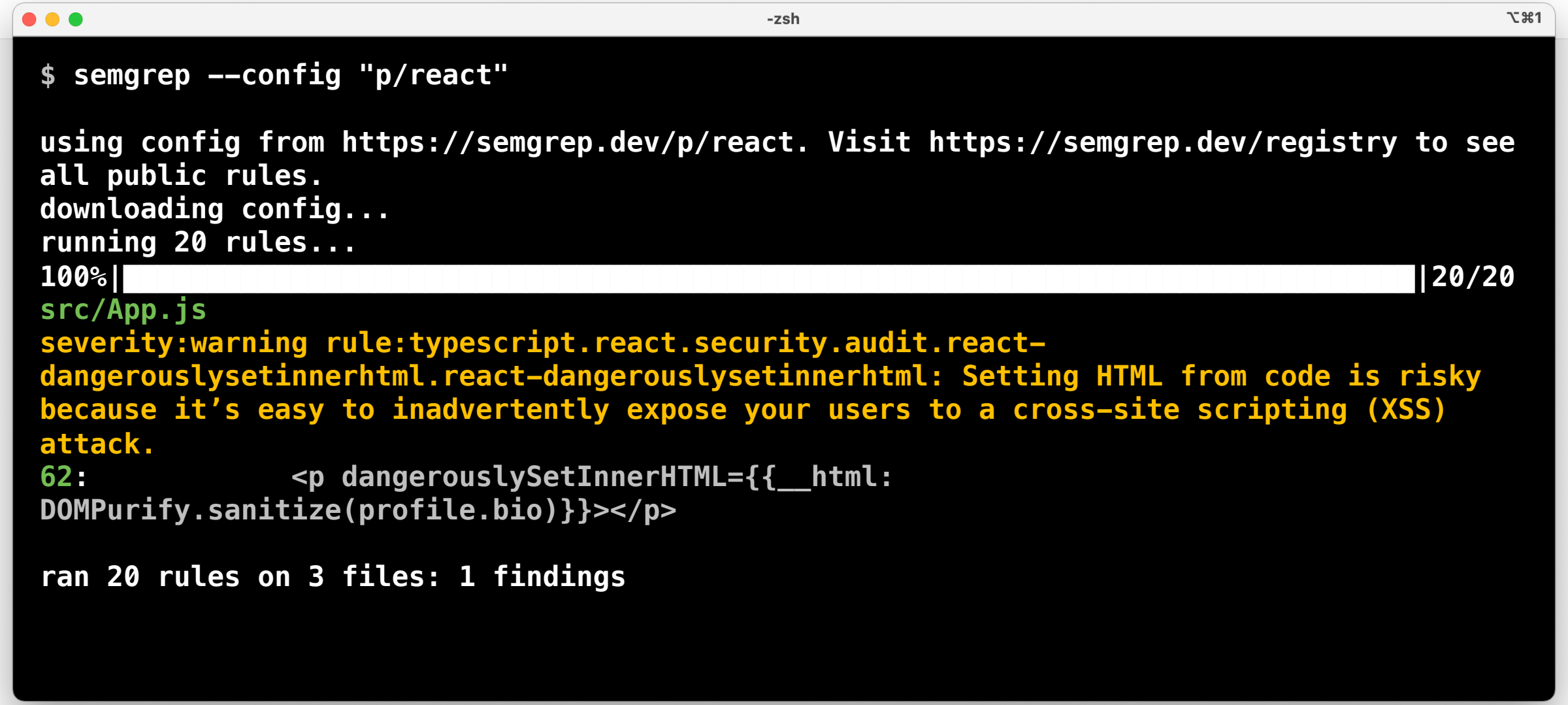
DOMPurify turns untrusted HTML in safe HTML, making it safe to include in the page

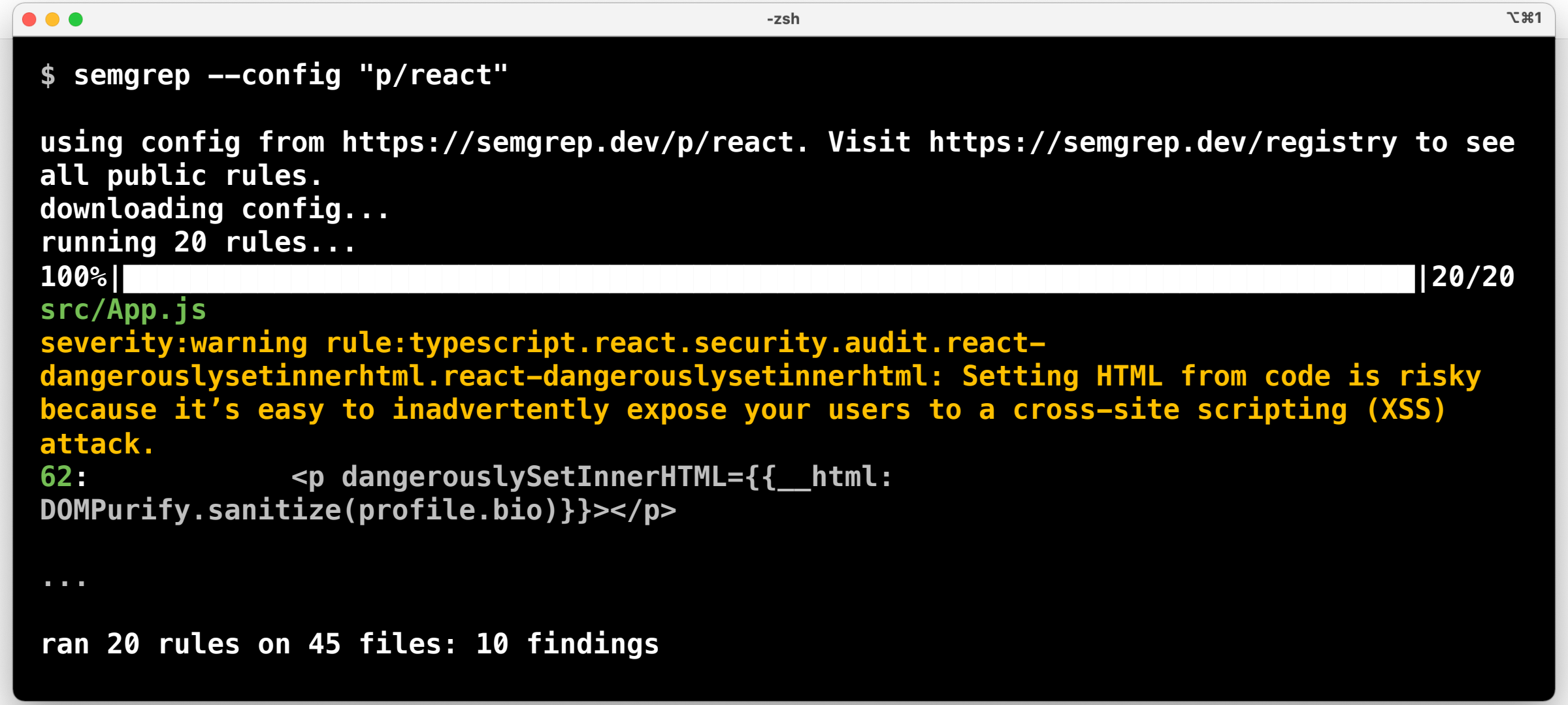
A review submitted by a malicious user

```
1  This restaurant is <b>highly recommended</b>. The
2  food is exquisite and the service is impeccable. <a
3  href="https://pics.example.com">Check out my story
4  here!</a><img src="none.png" onerror="alert('Go
5  there, now!')">
```

That wasn't too bad, now was it?


-zsh
⌥⌘1
$ semgrep --config "p/react"


-zsh
⌥⌘1
$ semgrep --config "p/react"
using config from https://semgrep.dev/p/react. Visit https://semgrep.dev/registry to see all public rules.
downloading config...
running 20 rules...
100%|20/20
src/App.js
severity:warning rule:typescript.react.security.audit.react-dangerouslysetinnerhtml.react-dangerouslysetinnerhtml: Setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack.
62: <p dangerouslySetInnerHTML={{__html: DOMPurify.sanitize(profile.bio)}}></p>
ran 20 rules on 3 files: 1 findings


-zsh
⌥⌘1
$ semgrep --config "p/react"
using config from https://semgrep.dev/p/react. Visit https://semgrep.dev/registry to see all public rules.
downloading config...
running 20 rules...
100%|20/20
src/App.js
severity:warning rule:typescript.react.security.audit.react-dangerouslysetinnerhtml.react-dangerouslysetinnerhtml: Setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack.
62: <p dangerouslySetInnerHTML={{__html: DOMPurify.sanitize(profile.bio)}}></p>
...
ran 20 rules on 45 files: 10 findings

Oh, crap!

A JSX template directly using dangerouslySetInnerHTML *(not recommended)*

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

A JSX template using a SafeHtml *component (recommended)*

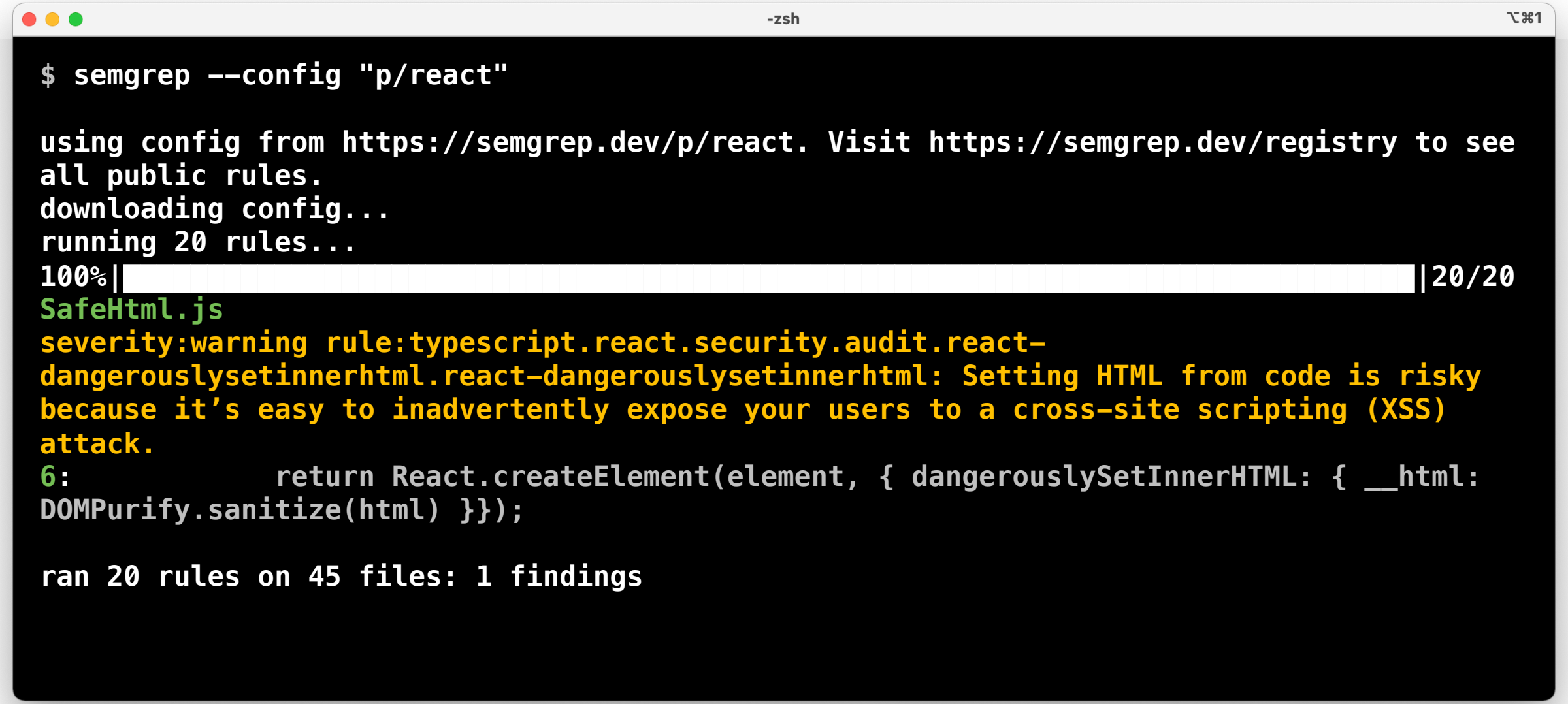
```
1  import SafeHtml from './SafeHtml';
2
3  return ( <div>
4    <h3>{ title }</h3>
5    <SafeHtml element="p" html={{review}}></SafeHtml>
6  </div>);
```

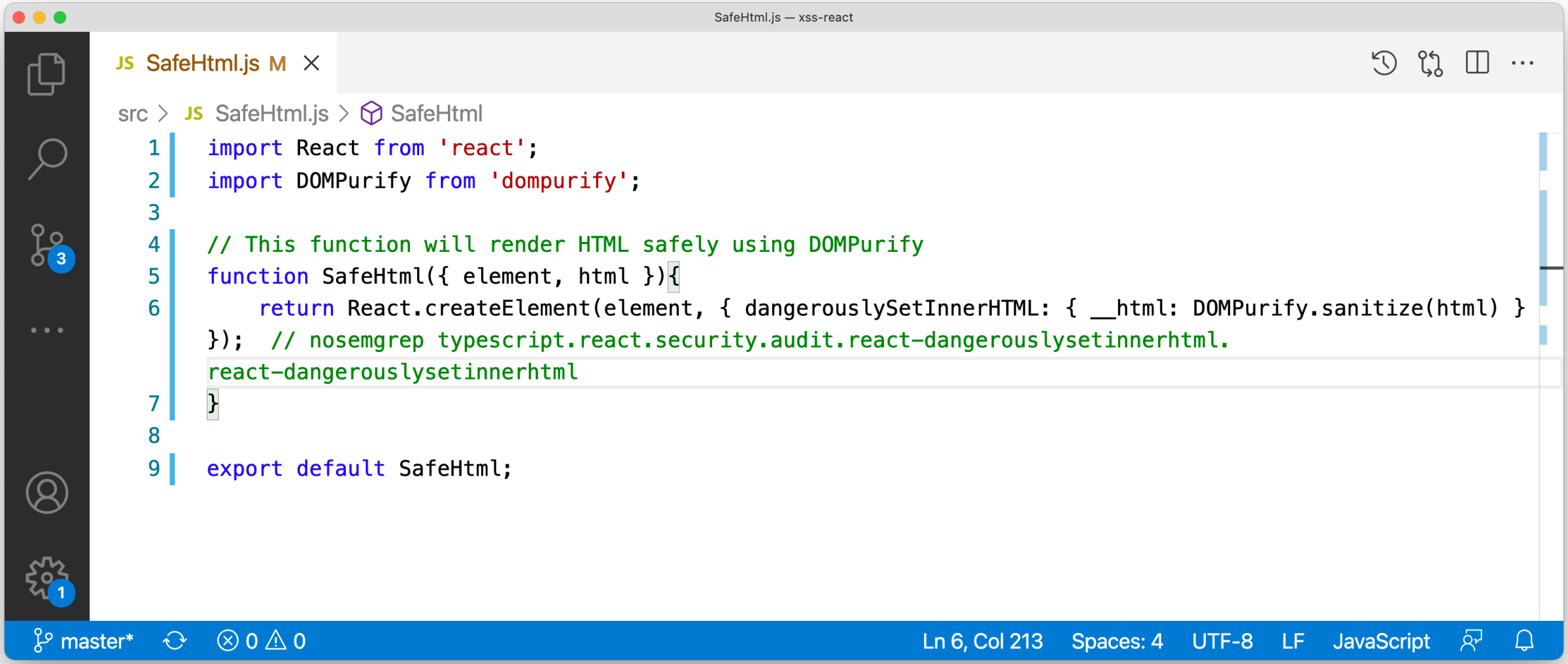
A JSX template using a SafeHtml *component (recommended)*

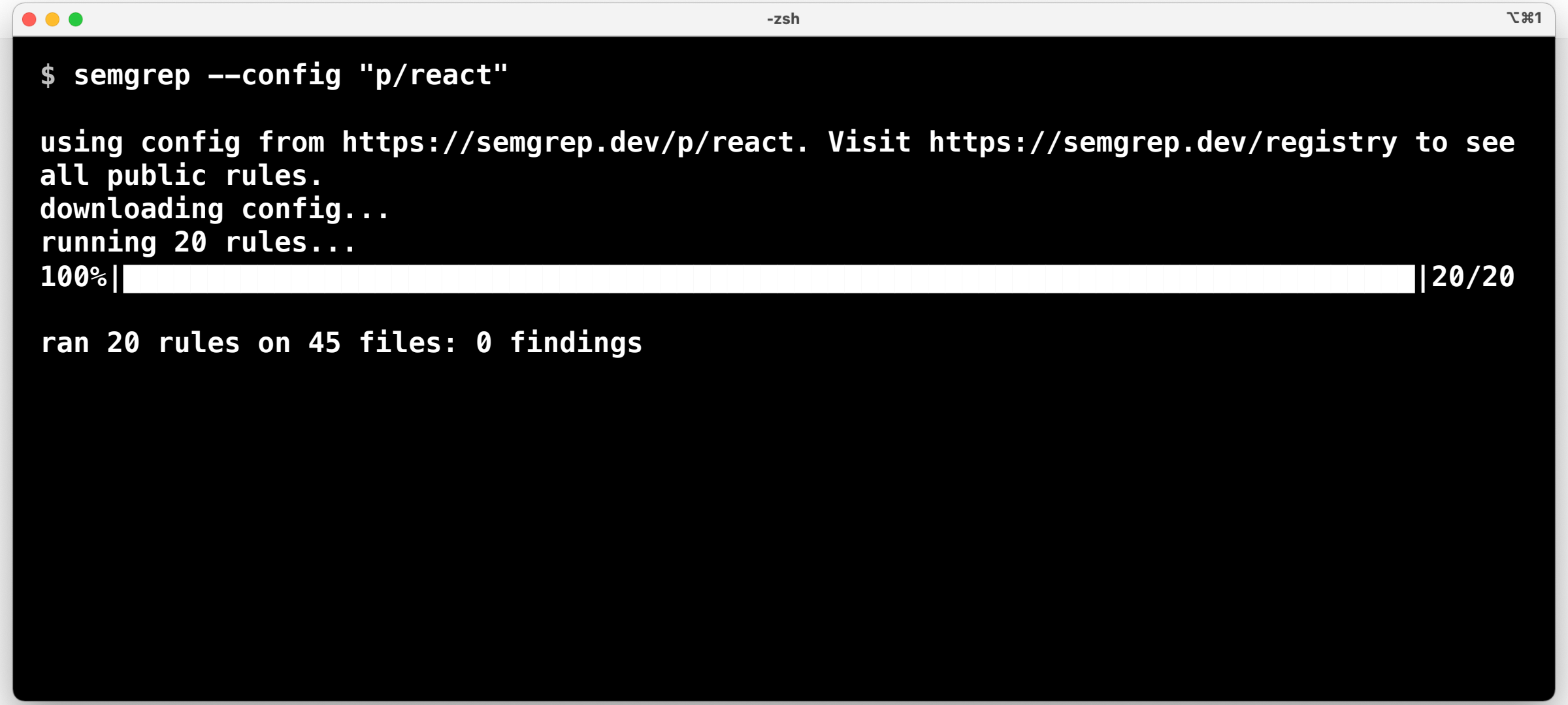
```
import SafeHtml from './SafeHtml';

return ( <div>
  <h3>{ title }</h3>
  <SafeHtml element="p" html={{review}}></SafeHtml>
</div>);
```

The SafeHtml *component*

```
import React from 'react';
import DOMPurify from 'dompurify';

// This function will render HTML safely using DOMPurify
function SafeHtml({ element, html }){
    return React.createElement(element, {
       dangerouslySetInnerHTML: { __html: DOMPurify.sanitize(html) }
    });
}
export default SafeHtml;
```


-zsh
⌥⌘1
$ semgrep --config "p/react"
using config from https://semgrep.dev/p/react. Visit https://semgrep.dev/registry to see all public rules.
downloading config...
running 20 rules...
100%|20/20
SafeHtml.js
severity:warning rule:typescript.react.security.audit.react-dangerouslysetinnerhtml.react-dangerouslysetinnerhtml: Setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack.
6: return React.createElement(element, { dangerouslySetInnerHTML: { __html: DOMPurify.sanitize(html) }});
ran 20 rules on 45 files: 1 findings


SafeHtml.js — xss-react
JS SafeHtml.js M
src > JS SafeHtml.js > SafeHtml
1 import React from 'react';
2 import DOMPurify from 'dompurify';
3
4 // This function will render HTML safely using DOMPurify
5 function SafeHtml({ element, html }){
6 return React.createElement(element, { dangerouslySetInnerHTML: { __html: DOMPurify.sanitize(html) }
}); // nosemgrep typescript.react.security.audit.react-dangerouslysetinnerhtml.
react-dangerouslysetinnerhtml
7 }
8
9 export default SafeHtml;
master*
0 0
Ln 6, Col 213
Spaces: 4
UTF-8
LF
JavaScript


-zsh
⌥⌘1
$ semgrep --config "p/react"
using config from https://semgrep.dev/p/react. Visit https://semgrep.dev/registry to see all public rules.
downloading config...
running 20 rules...
100%|20/20
ran 20 rules on 45 files: 0 findings

USING CODE HYGIENE FOR SECURITY SUCCESS

Developers should focus on development, not on fine-grained security rules

Encapsulate dangerous features in secure components

Use code analysis techniques to flag direct use of dangerous features

HTTPS://COURSES.PRAGMATICWEBSECURITY.COM

THANK YOU!

Follow me on Twitter to stay up to date on security resources and courses

@PhilippeDeRyck