Patterns and anti-patterns in software development

Dr. Philippe De Ryck

https://PragmaticWebSecurity.com

The road to AppSec hell

is paved with good intentions

Photo by Ybrayym Esenov on Unsplash

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


JUNIT

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
J1c2VyIjoiZTcyZDFhMjZmNDBlNGU4Nzk5NjciL
CJ0ZW5hbnQiOiJkOGNmM2ZhMzAxYTM0Yzk2ODUw
MmE3MDUxYmZkYzBhOCIsImlhdCI6MTYyMDE5MjY
0NDkxNCwiZXhwIjoxNjIwMTk2MjQ0OTE0fQ.bnd
YFgq1sHD-
vH8h1lARD8M0uZgoALThQu7CURkuSVs
```

The base64-encoded header and payload, along with the signature

The signature is crucial to ensure the integrity of the header and payload

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "user": "e72d1a26f40e4e879967",
  "tenant": "d8cf3fa301a34c968502a7051bfdc0a8",
  "iat": 1620192644914,
  "exp": 1620196244914
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  SuperSecretHMACKey
) ☐ secret base64 encoded
```

Apache Pulsar bug allowed account takeovers in certain configurations

Ben Dickson 02 June 2021 at 11:43 UTC
Updated: 02 June 2021 at 14:32 UTC

GitHub | Open Source Software | Secure Development

Software maintainers downplay real-world impact of JWT vulnerability

4 ...on/src/main/java/org/apache/pulsar/broker/authentication/AuthenticationProviderToken.java

```
@@ -172,9 +172,7 @@ private static String validateToken(final String token) throws AuthenticationExc
172       @SuppressWarnings("unchecked")
173       private Jwt<?, Claims> authenticateToken(final String token) throws AuthenticationException {
174           try {
175 -             Jwt<?, Claims> jwt = Jwts.parser()
176 -                     .setSigningKey(validationKey)
177 -                     .parse(token);
178
179               if (audienceClaim != null) {
180                   Object object = jwt.getBody().get(audienceClaim);
```

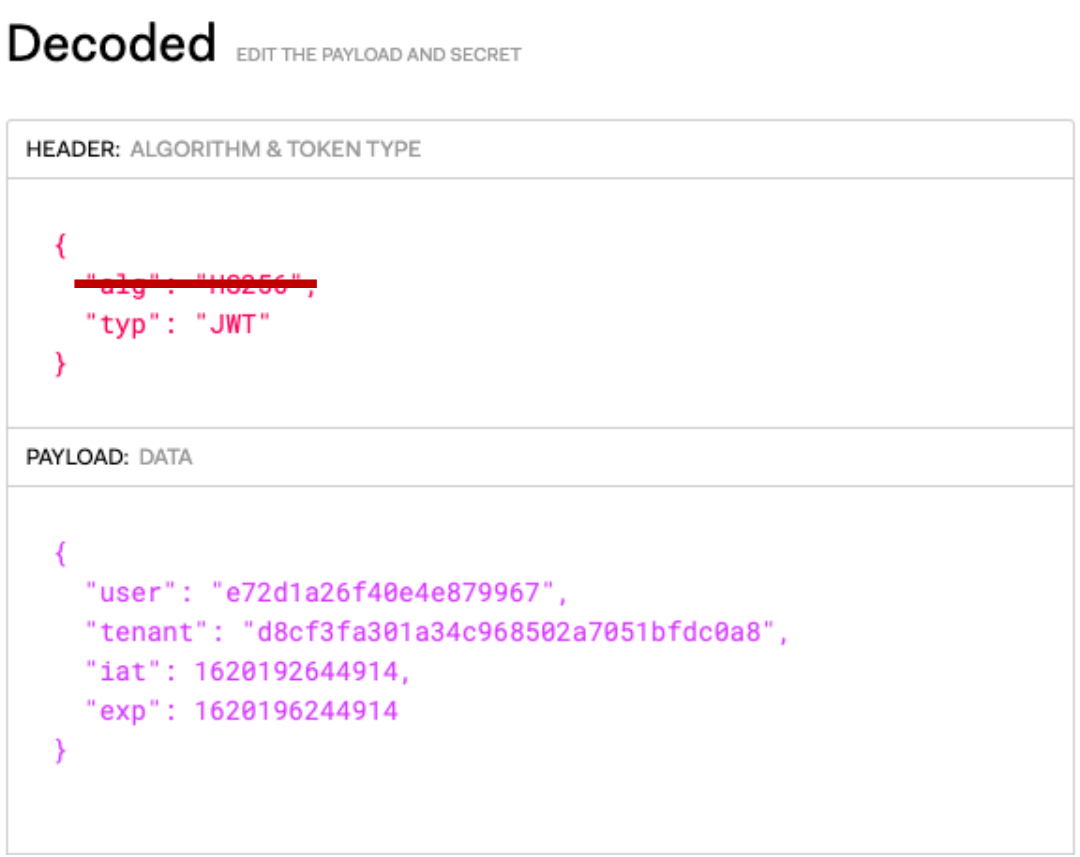
```
172       @SuppressWarnings("unchecked")
173       private Jwt<?, Claims> authenticateToken(final String token) throws AuthenticationException {
174           try {
175 +             Jwt<?, Claims> jwt = Jwts.parserBuilder().setSigningKey(validationKey).build()
                  .parseClaimsJws(token);
176
177               if (audienceClaim != null) {
178                   Object object = jwt.getBody().get(audienceClaim);
```

```
Jwts.parserBuilder()
    .setSigningKey(key)
    .build()
    .parse
```

```
parse(String jwt) : Jwt     JwtParser.parse(String jwt) : Jwt
parse(String jwt, JwtHandler<T> handler) : T
parseClaimsJws(String claimsJws) : Jws<Claims>
parseClaimsJwt(String claimsJwt) : Jwt<Header,Claims>
parsePlaintextJws(String plaintextJws) : Jws<String>
parsePlaintextJwt(String plaintextJwt) : Jwt<Header,…
```

```java
/**
 * Parses the specified compact serialized JWT string based on the builder's current configuration state and
 * returns the resulting JWT or JWS instance.
 * <p>
 * <p>This method returns a JWT or JWS based on the parsed string.  Because it may be cumbersome to determine if it
 * is a JWT or JWS, or if the body/payload is a Claims or String with {@code instanceof} checks, the
 * {@link #parse(String, JwtHandler) parse(String,JwtHandler)} method allows for a type-safe callback approach that
 * may help reduce code or instanceof checks.</p>
 *
 * @param jwt the compact serialized JWT to parse
 * @return the specified compact serialized JWT string based on the builder's current configuration state.
 * @throws MalformedJwtException    if the specified JWT was incorrectly constructed (and therefore invalid).
 *                                  Invalid
 *                                  JWTs should not be trusted and should be discarded.
 * @throws SignatureException       if a JWS signature was discovered, but could not be verified.  JWTs that fail
 *                                  signature validation should not be trusted and should be discarded.
 * @throws ExpiredJwtException      if the specified JWT is a Claims JWT and the Claims has an expiration time
 *                                  before the time this method is invoked.
 * @throws IllegalArgumentException if the specified string is {@code null} or empty or only whitespace.
 * @see #parse(String, JwtHandler)
 * @see #parsePlaintextJwt(String)
 * @see #parseClaimsJwt(String)
 * @see #parsePlaintextJws(String)
 * @see #parseClaimsJws(String)
 */
Jwt parse(String jwt) throws ExpiredJwtException, MalformedJwtException, SignatureException, IllegalArgumentException;
```


https://github.com/jwtk/jjwt/blob/master/api/src/main/java/io/jsonwebtoken/JwtParser.java


Decoded
EDIT THE PAYLOAD AND SECRET
HEADER: ALGORITHM & TOKEN TYPE
{
"alg": "HS256",
"typ": "JWT"
}
PAYLOAD: DATA
{
"user": "e72d1a26f40e4e879967",
"tenant": "d8cf3fa301a34c968502a7051bfdc0a8",
"iat": 1620192644914,
"exp": 1620196244914
}

alg: none

alg: none

eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0
.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ
.

Include common pitfalls in your test scenarios

Test your applications to ensure JWTs with "alg:none" are rejected.

AppSec is too hard!

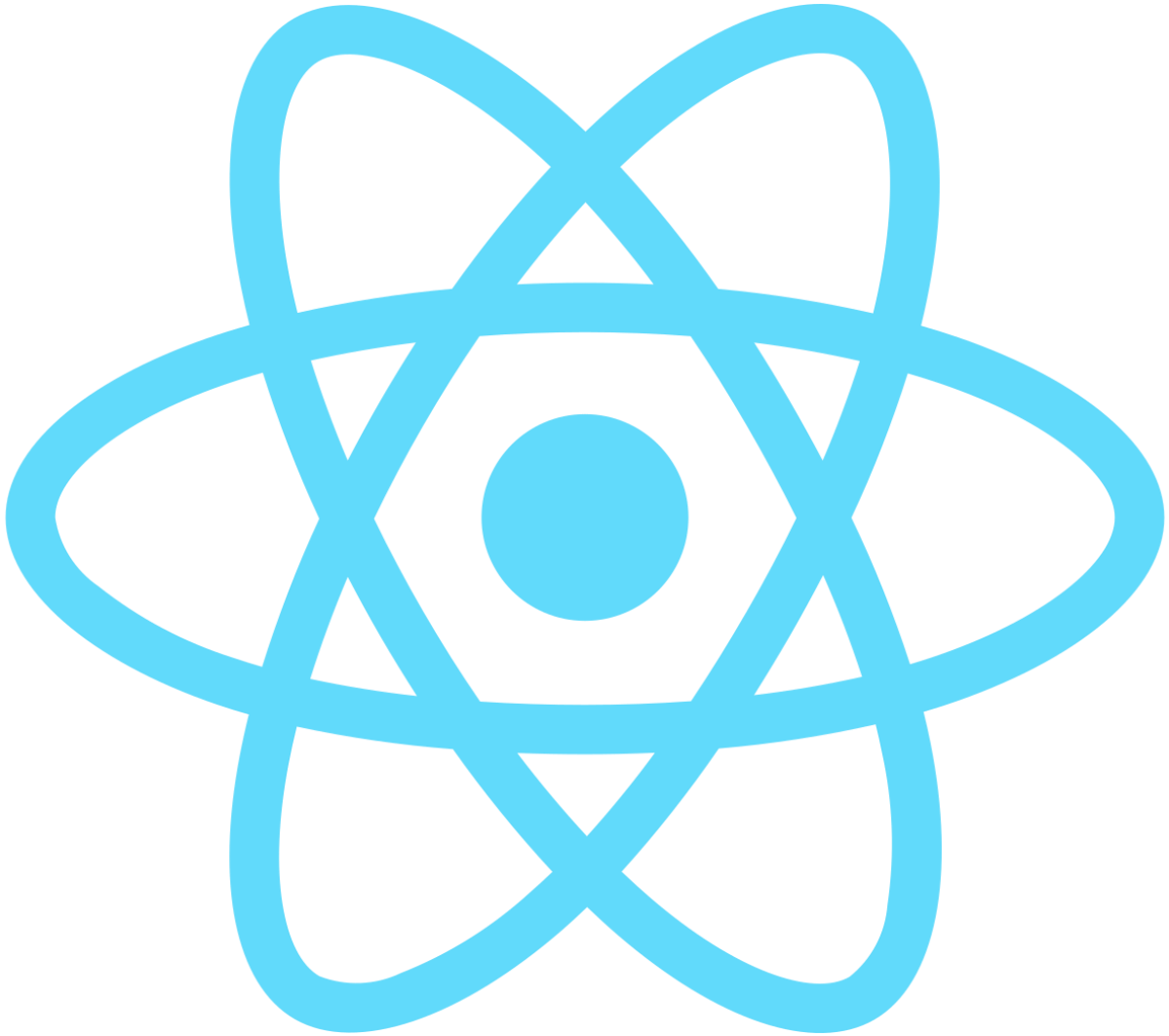
```
Jwts.parserBuilder()
    .setSigningKey(key)
    .build()
    .parse
```

```
parse(String jwt) : Jwt     JwtParser.parse(String jwt) : Jwt
parse(String jwt, JwtHandler<T> handler) : T
parseClaimsJws(String claimsJws) : Jws<Claims>
parseClaimsJwt(String claimsJwt) : Jwt<Header,Claims>
parsePlaintextJws(String plaintextJws) : Jws<String>
parsePlaintextJwt(String plaintextJwt) : Jwt<Header,…
```

```
Claims claims =
  Security.verifyAuthenticationToken(token);
```

Encapsulate security behavior in libraries

Offering the right abstractions absolves developers of the responsibility to apply detailed secure coding guidelines



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.

dangerouslySetInnerHTML

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

Avoid accidental misuse of dangerous features

Explicitly mark dangerous application features as dangerous to raise developer awareness

Photo by Stephen Andrews on Unsplash

dangerouslySetInnerHTML

`dangerouslySetInnerHTML` is React's replacement for using `innerHTML` in the browser DOM. In general, setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack. So, you can set HTML directly from React, but you have to type out `dangerouslySetInnerHTML` and pass an object with a `__html` key, to remind yourself that it's dangerous. For example:

```
function createMarkup() {
  return {__html: 'First &middot; Second'};
}

function MyComponent() {
  return <div dangerouslySetInnerHTML={createMarkup()} />;
}
```


https://reactjs.org/docs/dom-elements.html

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
```

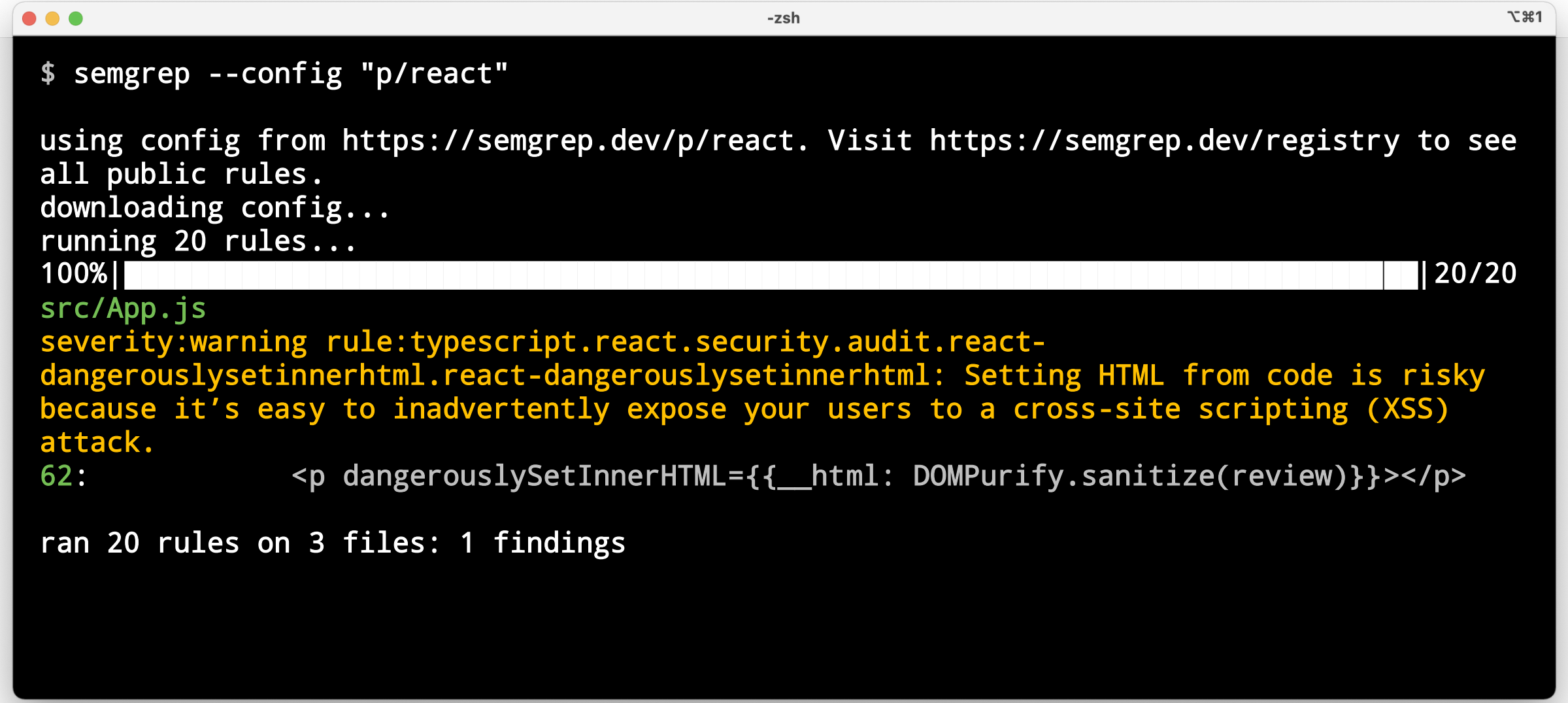
```
113
114         if (text) {
115           return (
116 +           <div className="text">
117 +             <MessageBody text={text} />
118 +           </div>
119           );
120         }
121
```

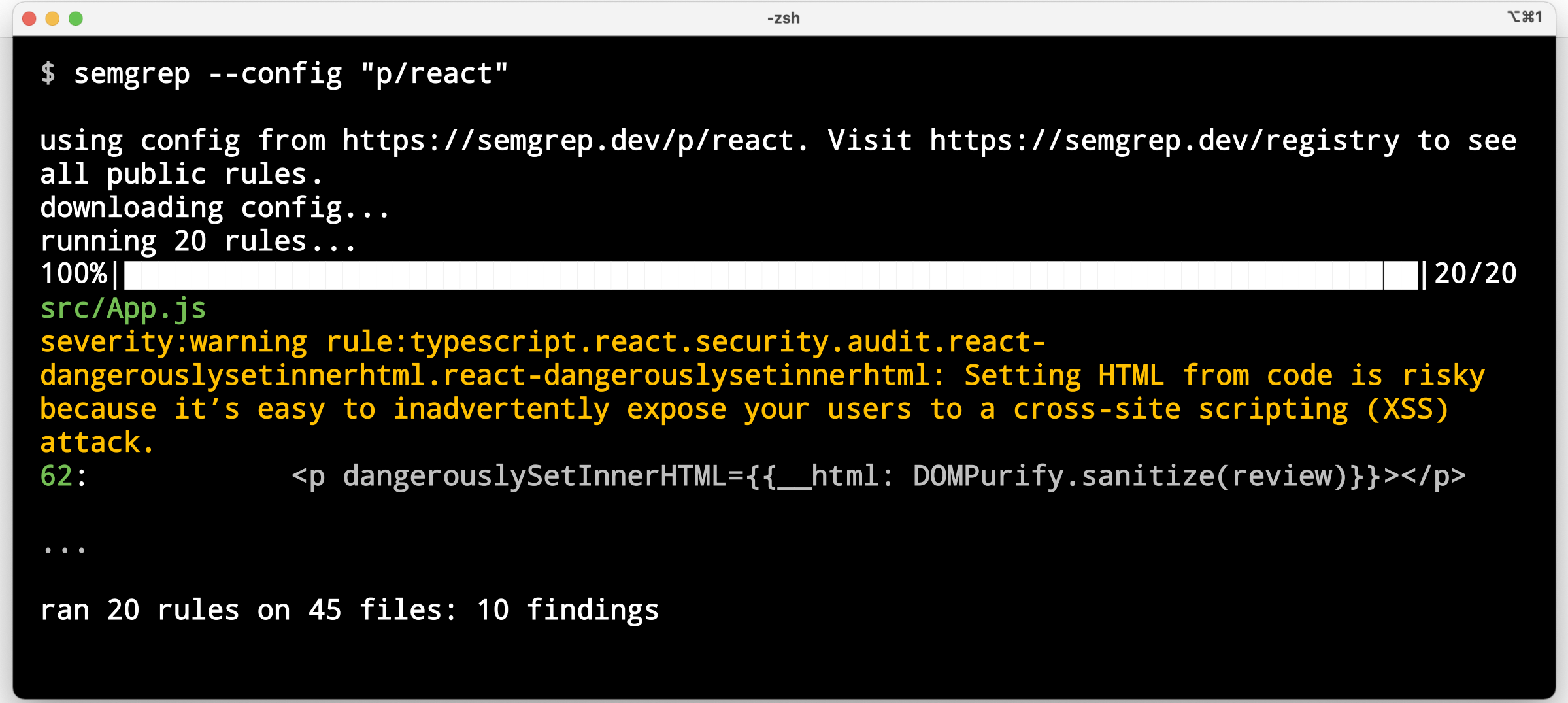

https://github.com/signalapp/Signal-Desktop/commit/4e5c8965ff72576a9e20850dd30d9985f4073192#diff-8bba204372da85d8cceed81278b7eec

Marking things as dangerous is not enough

Explicitly marking dangerous features prevents accidental mis-use, but does not magically enable developers to use the feature securely


-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(review)}}></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(review)}}></p>
...
ran 20 rules on 45 files: 10 findings

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>);
```

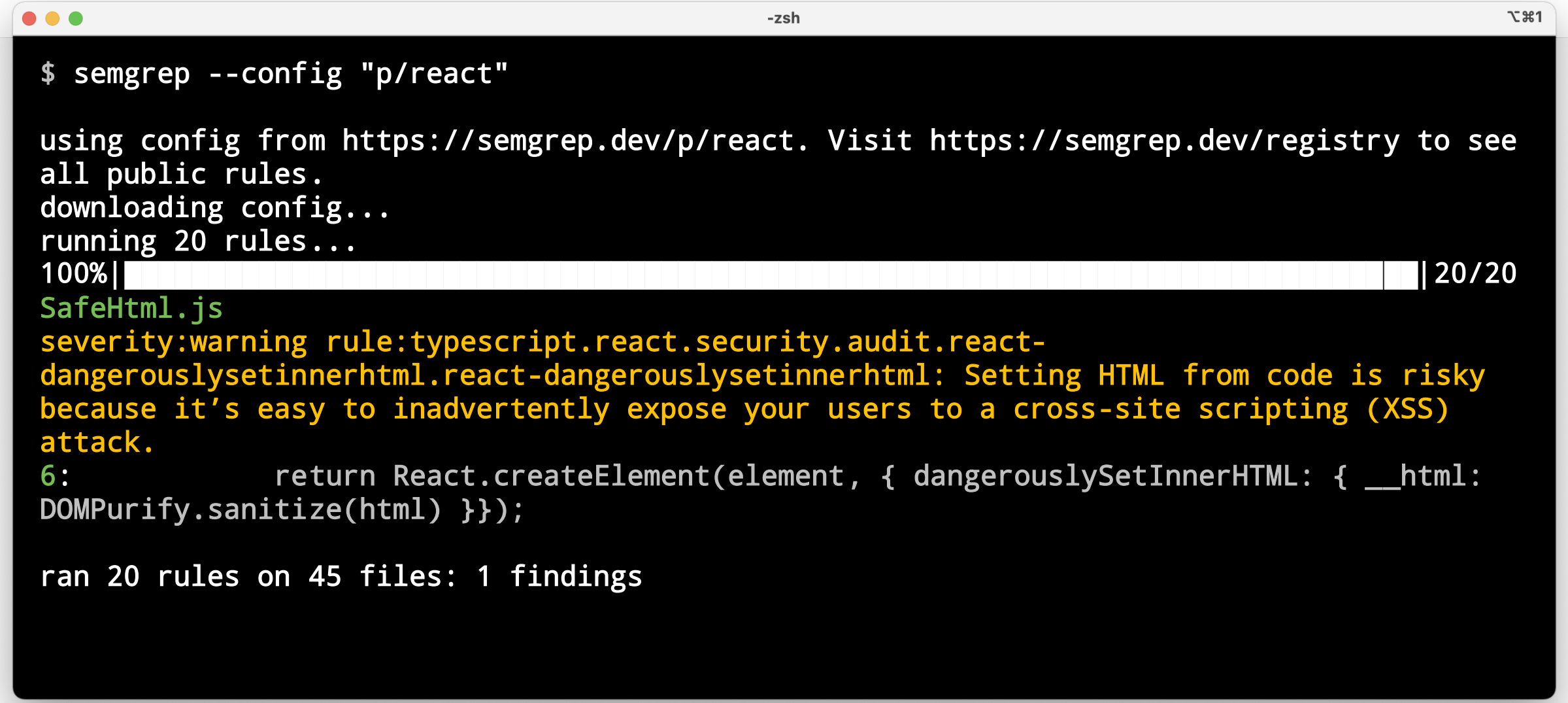
The SafeHtml *component*

```
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;
```

Encapsulate security behavior in libraries

Offering the right abstractions absolves developers of the responsibility to apply detailed secure coding guidelines


-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

@PhilippeDeRyck


-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

Code hygiene and encapsulation make it easier to apply security best practices at scale

THANK YOU!

LEARN MORE ABOUT REACT/API SECURITY BEST PRACTICES ...

HTTPS://COURSES.PRAGMATICWEBSECURITY.COM