PATTERNS AND ANTI-PATTERNS IN SOFTWARE DEVELOPMENT

DR. PHILIPPE DE RYCK

https://Pragmatic Web Security.com
The road to AppSec hell is paved with good intentions.
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
The base64-encoded header and payload, along with the signature

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

Software maintainers downplay real-world impact of JWT vulnerability
@SuppressWarnings("unchecked")
private Jwt<?, Claims> authenticateToken(final String token) throws AuthenticationException {
    try {
        Jwt<?, Claims> jwt = Jwts.parser()
            .setSigningKey(validationKey)
            .parse(token);
        if (audienceClaim != null) {
            Object object =
                jwt.getBody().get(audienceClaim);
        }
    }
}
Jwts.parserBuilder()
  .setSigningKey(key)
  .build()
  .parse

parse(String jwt) : Jws
  JwtParser.parse(String jwt) : Jws
  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,...
/**
 * Parses the specified compact serialized JWT string based on the builder's current configuration state and
 * returns the resulting JWT or JWS instance.
 * <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;
<table>
<thead>
<tr>
<th>HEADER: ALGORITHM &amp; TOKEN TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;alg&quot;: &quot;HS256&quot;,</td>
</tr>
<tr>
<td>&quot;typ&quot;: &quot;JWT&quot;</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAYLOAD: DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>&quot;user&quot;: &quot;e72d1a26f40e4e879967&quot;,</td>
</tr>
<tr>
<td>&quot;tenant&quot;: &quot;d8cf3fa301a34c968502a7051bfdc8a8&quot;,</td>
</tr>
<tr>
<td>&quot;iat&quot;: 162819264914,</td>
</tr>
<tr>
<td>&quot;exp&quot;: 162819524914</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
alg: none

eyJhbGciOiJub25lIiwidHlwIjoiSldUIn0

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ

gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ
INCLUDE COMMON PITFALLS IN YOUR TEST SCENARIOS

Test your applications to ensure JWTs with "alg:none" are rejected.
AppSec is too hard!
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 review submitted by a malicious user

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

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
The greatest things you learn from traveling

The great things on earth traveling teaches us by example. Here are some of the marvelous 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 allows you to appreciate each and every single one of the differences while you become more fluid.

@PhilippeDeRyck
dangerouslySetInnerHTML
**A JSX template to render user-provided HTML with a major vulnerability**

```jsx
return (<div>
  <h3>{title}</h3>
  <p dangerouslySetInnerHTML={{__html: review}}></p>
</div>);
```

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

---

**A JSX template to render user-provided HTML using DOMPurify**

```jsx
import DOMPurify from 'dompurify';

return (<div>
  <h3>{title}</h3>
  <p dangerouslySetInnerHTML={{__html: DOMPurify.sanitize(review)}}></p>
</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
CAUTION

DANGER
FALL
HAZARD

ATTENTION

RISQUE
DE
CHUTE

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:

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

function MyComponent() {
  return <div dangerouslySetInnerHTML={createMarkup()} />
}
```
@@ -111,7 +113,9 @@ export class Quote extends React.Component<Props, {}> {

    if (text) {
        return (
-            <div className="text" dangerouslySetInnerHTML={{
+            <div className="text">
+            <MessageBody text={text} />
+        </div>

+        );
        }

    }

@@ -118 +120 }
Explicitly marking dangerous features prevents accidental misuse, but does not magically enable developers to use the feature securely.
$ semgrep --config "p/react"
$ semgrep --config "p/react"


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
$ semgrep --config "p/react"


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)

```javascript
import DOMPurify from 'dompurify';

return (<div>
    <h3>{title}</h3>
    <p dangerouslySetInnerHTML={{__html: DOMPurify.sanitize(review)}}></p>
</div>);
```

A JSX template using a SafeHtml component (recommended)

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

return (<div>
    <h3>{title}</h3>
    <SafeHtml element="p" html={{review}}></SafeHtml>
</div>);
```
A JSX template using a SafeHtml component (recommended)

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

function SafeHtml({ element, html }){
  return React.createElement(element, {
    dangerouslySetInnerHTML: { __html: DOMPurify.sanitize(html) }
  });
}

export default SafeHtml;
```

The SafeHtml component

```javascript
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
$ semgrep --config "p/react"

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
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;
$ semgrep --config "p/react"


downloading config...

running 20 rules...

100% | 20/20

ran 20 rules on 45 files: 0 findings
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