AVOIDING XSS IN REACT APPLICATIONS

React is a popular framework for building a modern JS frontend application. By default, data binding in React happens in a safe way, helping developers to avoid Cross-Site Scripting (XSS) issues. However, data used outside of simple data bindings often results in dangerous XSS vulnerabilities. This cheat sheet gives an overview of secure coding guidelines for React.

SIMPLE DATA BINDING

By default, React prevents data to be seen as code. The default data binding mechanism does not cause HTML injection attacks. When possible, always use **{}** for data binding.

- ☐ Use **{}** to place simple data inside HTML elements

```
return ( <div>{ data }</div> );
```

- ☐ Use variables to assign values to keys in the properties

```
<li style={{ color: data }}>
```

RENDERING BENIGN HTML

Simple data binding does not work when the data needs to be rendered as HTML. Without adequate security, rendering HTML causes XSS vulnerabilities. Always ensure the output is properly sanitized.

- HTML output requires using **dangerouslySetInnerHTML**

```
return ( <div dangerouslySetInnerHtml=
  {{__html: data }}></div> );
```

Without proper sanitization, this pattern is extremely dangerous

- ☐ Install ***DOMPurify***, a JS HTML sanitizer, as a dependency

```
npm install dompurify
```

- ☐ Load ***DOMPurify***, a JS HTML sanitizer, in the React app

```
const createDOMPurify = require('dompurify');
const purify = createDOMPurify(window);
```

- ☐ Always sanitize data being used as HTML output

```
return ( <div dangerouslySetInnerHtml=
  {{__html: purify.sanitize(data)}}></div> );
```

DOMPurify ensures that the output only contains benign HTML

- Setup linting rules to detect **dangerouslySetInnerHTML**

USING HTML-TO-REACT PARSERS

Libraries such as ***html-react-parser*** enable the parsing of HTML into React elements. These libraries avoid certain XSS attack vectors, but do not offer a reliable security mechanism. Always sanitize data with ***DOMPurify***.

- ☐ Avoid relying on HTML parsing libraries for security

```
return (<div>{ ReactHtmlParser(data) }</div>);
```

Without proper sanitization, this pattern creates XSS issues

HANDLING DYNAMIC URLS

URLs derived from untrusted input often cause XSS through obscure features, such as the **javascript:** scheme or **data:text/html** scheme. Dynamic URLs need to be vetted for security before they are used.

- ☐ Never allow unvetted data in an **href** or **src** attribute

```
return ( <a href={ data }>Click me!</a> );
```

- ☐ If possible, hardcode the *scheme* / *host* / *path* separator

```
var url = "https://example.com/" + data
```

This pattern guarantees a fixed destination for this URL

- ☐ Use a URL sanitization library to sanitize untrusted URLs

Re-using Angular's URL sanitization is the most secure solution

- ☐ Use ***DOMPurify*** to output HTML with dynamic URLs

DOMPurify removes HTML attributes that contain unsafe URLs

ACCESSING NATIVE DOM ELEMENTS

Traditional web applications suffer from DOM-based XSS when they insecurely insert data into the DOM. React applications can create similar vulnerabilities by insecurely accessing native DOM elements.

- ☐ Avoid DOM manipulation through insecure APIs

innerHTML *and* **outerHTML** *often cause DOM-based XSS*

- ☐ Scan your codebase for references to native DOM elements

React's **createRef** *function exposes DOM elements*

ReactDOM's **findDOMNode** *function exposes DOM elements*

- ☐ When DOM manipulation cannot be avoided, use safe APIs

E.g., **document.createElement** *instead of* **innerHTML**

INSERTING DYNAMIC JSON DATA

The insecure serialization of JSON data often results in XSS vulnerabilities. It allows the attacker to control React object properties or to break out of the JavaScript environment.

- ☐ Do not use an untrusted object as the props of an element

Always assign individual property values instead of a full object

- ☐ Do not use **stringify** to put JSON data in a script context

Secure serialization avoids confusion between JS and HTML [1]

[1] https://bit.ly/3aX3pcq

Looking for applicable advice on building secure React apps?

Reach out to discuss a practical training course on current best practices

COURSES@PRAGMATICWEBSECURITY.COM

Hands-on training course