Cheat Sheet: 10 React Security Best Practices

snyk

1. Default XSS Protection with Data Binding

Use default data binding with curly braces {} and React will automatically escape values to protect against XSS attacks.
Note that this protection only occurs when rendering textContent and not when rendering HTML attributes.

- Use JSX data binding syntax {} to place data in your elements. Ex. `<div>{data}</div>`.
- Avoid dynamic attribute values without custom validation. Don't use `<form action={data}>...`

2. Dangerous URLs

URLs can contain dynamic script content via javascript: protocol urls. Validate URLs before use.

- Use validation to assure your links are http or https to avoid javascript: URL based script injection. Achieve URL validation using a native URL parsing function then match the parsed protocol property to an allow list.

3. Rendering HTML

It is possible to insert HTML directly into rendered DOM nodes using dangerouslySetInnerHTML. Any content inserted this way must be sanitized beforehand.

- Use a sanitization library like DOMPurify on any values before placing them into the dangerouslySetInnerHTML prop.

```
import purify from "dompurify";

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

4. Direct DOM Access

Accessing the DOM to inject content into DOM nodes directly should be avoided. Use dangerouslySetInnerHTML if you must inject HTML and sanitize it before injecting it using DOMPurify.

- Avoid using refs and findDomNode() to access rendered DOM elements to directly inject content via innerHTML and similar properties and methods.

5. Server-side Rendering

Data binding will provide automatic content escaping when using server-side rendering functions like ReactDOMServer.renderTo-String() and ReactDOMServer.renderToStaticMarkup().

- Avoid concatenating strings onto the output of renderTo String() and renderToStaticMarkup() before sending the strings to the client for hydration or rendering.

6. Known Vulnerabilities in Dependencies

Some versions of third-party components might contain security issues. Check your dependencies and update when better versions become available.

- Use a tool like the free Snyk cli to check for vulnerabilities.
- Automatically fix vulnerabilities with Snyk by integrating with your source code management system to receive automated fixes.

```
$ npx snyk test
```

7. JSON State

It is common to send JSON data along with server-side rendered React pages. Always escape < characters with a benign value to avoid injection attacks.

- Avoid unescaped HTML significant values in JSON state objects.

```
<script>
        // WARNING: See the following for security
issues around embedding JSON in HTML:
        // https://redux.js.org/recipes/server-ren-
dering/#security-considerations
        window.__PRELOADED_STATE__ = ${JSON.stringi-
fy(preloadedState).replace(
          /</g,
          '\\u003c')}
</script>
```

8. Vulnerable Versions of React

The React library has had a few high severity vulnerabilities in the past, so it is a good idea to stay up-to-date with the latest version.

- Avoid vulnerable versions of the react and react-dom by verifying that you are on the latest version using npm outdated to see the latest versions.
- Use Snyk to automatically update to new versions when vulnerabilities exist in the versions you are using.

9. Linters

Install Linters configurations and plugins that will automatically detect security issues in your code and offer remediation advice.

- Use the ESLint React security config to detect security issues in our code base.
- Configure a pre-commit hook that fails when security related linter issues are detected using a library like husky.

10. Dangerous Library Code

Library code is often used to perform dangerous operations like directly inserting HTML into the DOM. Review library code manually or with linters to detect unsafe usage of React's security mechanisms.

- Avoid libraries that do use dangerouslySetInnerHTML, innerHTML, unvalidated URLs or other unsafe patterns.
- Use security linters on your node_modules folder to detect unsafe patterns in your library code.

Ron Perris
Developer Advocate at Snyk

Liran Tal
Node.js Security WG & Developer Advocate at Snyk