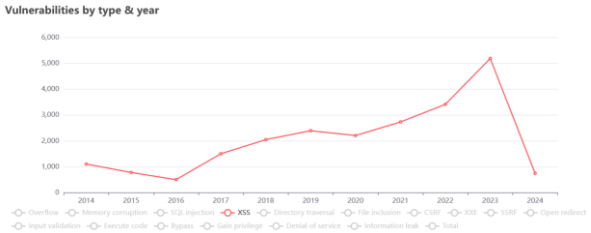
RUMBLE IN THE (BROWSER) JUNGLE
XSS VS. SPA

confoo
#confoo

Christian Wenz
info@christianwenz.de
@chwenz

XSS

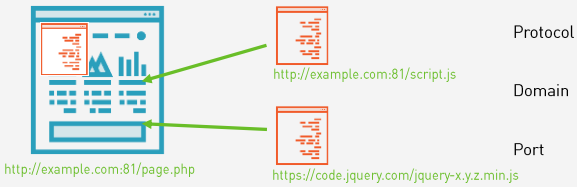


- https://www.cvedetails.com/vulnerabilities-by-types.php

Countermeasures

- Escape output: < > " ' &
- PHP: `htmlspecialchars()`
- ASP.NET Core: @, `HttpUtility.HtmlEncode()`
- But that's only the server

Same-Origin Policy (SOP)



4

Angular vs. XSS

- Angular trusts no-one.
  - Except for templates
- HTML encoding: `<p>{{property}}</p>`
- No encoding: `<p [innerHTML]="property"></p>`
- Special case URLs
  - `<a [href]="url"></a>`
  - `<iframe [src]="url"></iframe>`

5

Angular vs. XSS (2)

- Handling „unsafe" values
  - Sanitize them with `DomSanitizer.sanitize()`
  - Use them nevertheless with `bypassSecurityTrustHtml()` and others

React vs. XSS

- Automated escaping in JSX
- HTML encoding: `<p>{SomeValue}</p>`
  - Not sufficient in all cases (e.g. URLs)!
- No encoding:
  - `<p dangerouslySetInnerHTML={{__html: SomeValue}}></p>`
  - `React.createElement()`
  - Server templates

7

Vue vs. XSS

- HTML encoding: `<p>{{property}}</p>`
- Same for attribute bindings: `<input :value="property">`
- No escaping when binding HTML: `<p v-html="property"></p>`
- No escaping when using JSX: `<p innerHTML={ this.property }></p>`
- No escaping when using a render function: `h("p", { innerHTML: this.property })`

Content Security Policy

```
Content-Security-Policy:
       default-src 'self';
       img-src 'self' https://static.example.com;
```

Direktives: Loading Resources

- default-src
- child-src
- connect-src
- font-src
- img-src
- media-src
- object-src
- script-src
- style-src

Enable Inline Code

- Via token

```
script-src 'nonce-abc123def'
<script nonce="abc123def"> /* code */ </script>
```

- Via hash

```
script-src 'sha256-KF/49rpKwd...'
<script> /* code */ </script>
```

11

DOMPurify

- Open source sanitizer library
- Runs on the client, in the DOM – fast, efficient, and „close to the source"
- API: `DOMPurify.sanitize(input)`
- Removes select HTML element, and most attributes
- Highly configurable
- https://github.com/cure53/DOMPurify

12

HTML Sanitizer API

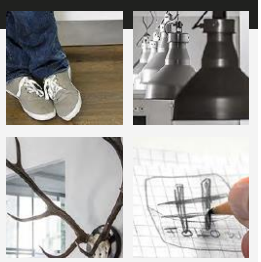
- https://wicg.github.io/sanitizer-api/
- Cleans HTML (certain tags, most attributes)
- Highly configurable
- Parts of the API are implemented in Chrome, Edge, and in Firefox (feature flag, different API)
- However API changes made Chrome temporarily drop support
  - https://developer.chrome.com/blog/sanitizer-api-deprecation

Trusted Types

- Do not assign strings to *innerHTML* (and other attack vectors)
- `Content-Security-Policy: require-trusted-types-for 'script'`
- ```
  trustedTypes.createPolicy(
     "default",
     {
       createHTML: s => DomPurify.sanitize(
                           s, { RETURN_TRUSTED_TYPE: true })
     }
   );
  ```

Thank You!

- Questions?
- info@christianwenz.de
- @chwenz


  ```