TranSPArent

Taint-Style Vulnerability Detection in General Single Page Applications through Automated Framework Abstraction

S.S. Diwangkara, Yinzhi Cao

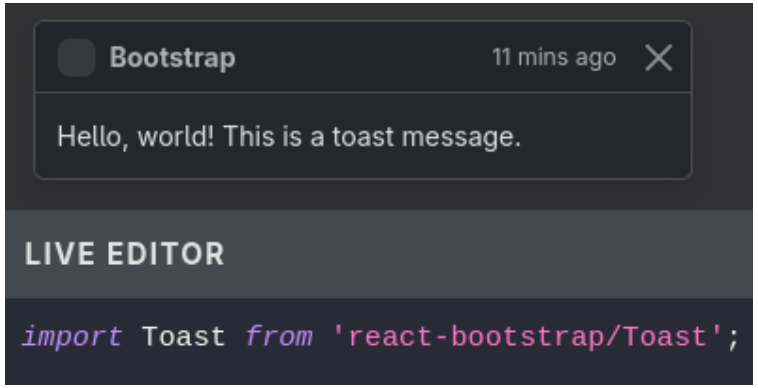
JOHNS HOPKINS UNIVERSITY

Single Page Applications (SPAs)

The *de facto* framework of modern web applications, millions of weekly download (npm)

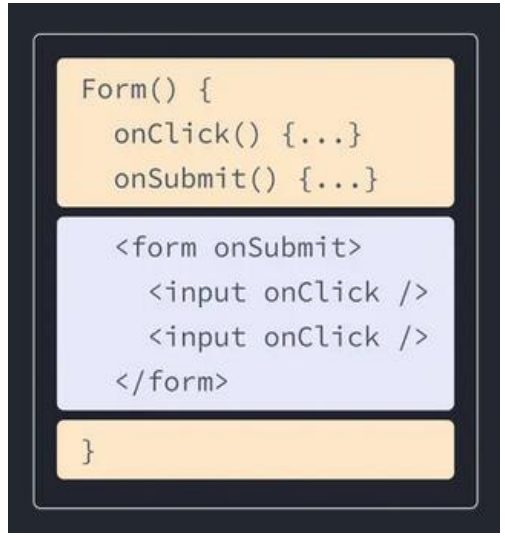
Rich Client Functionality

- Variable binding
- State management



High-Level Component

- Composable & reusable
- Rich library ecosystem



Expressive Template

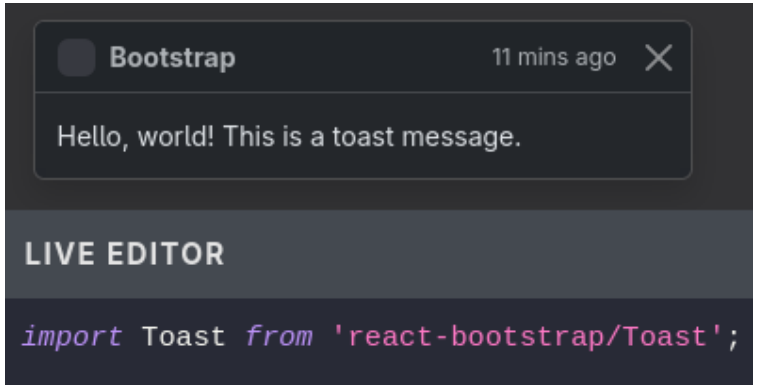
- HTML-JS embedding
- Contained in one file

Single Page Applications (SPAs)

The *de facto* framework of modern web applications, millions of weekly download (npm)

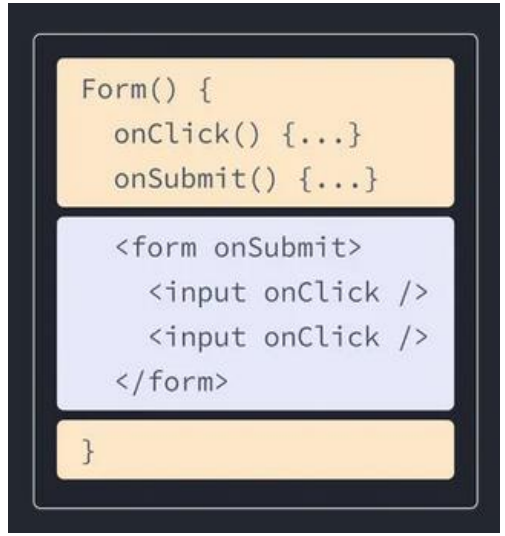
Rich Client Functionality

- Variable binding
- State management



High-Level Component

- Composable & reusable
- Rich library ecosystem



Expressive Template

- HTML-JS embedding
- Contained in one file

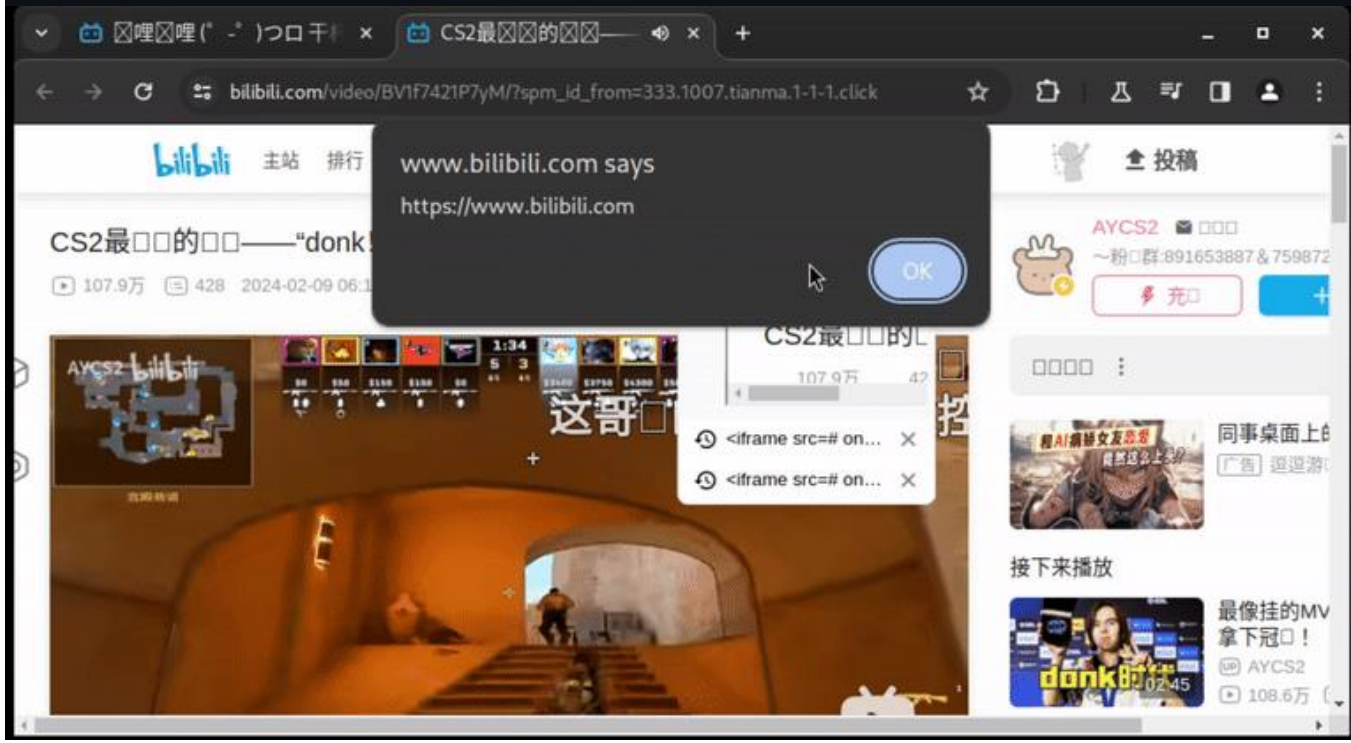
But is it secure?

SPAs Could be Vulnerable Too

```
Vue.extend({
  render: h => {
    const content = h('div', {
      domProps: {
        innerHTML: /* html */ `<em class="suggest-highlight">${input}</em>`,
      },
    })
    return content
  },
}),
```

Example: BiliBili-Evolved

- A script to enhance search result
- Vulnerable to reflected XSS from the search response
- State-of-the-art tool (CodeQL) is unable to detect the vulnerability due to missing sink definition



Detection Challenges

- **Sink diversity**
  - Many different frameworks, each with their own syntax
  - Multiple syntaxes in the same framework (JavaScript and HTML-like syntax)
- **Taint-rule diversity**
  - Dynamic features hinder taint-flow detection, as with many JavaScript analysis
  - Scalability problem in whole program analysis

Detection Challenges

- **Sink diversity**
  - Many different frameworks, each with their own syntax
  - Multiple syntaxes in the same framework (JavaScript and HTML-like syntax)
- **Taint-rule diversity**
  - Dynamic features hinder taint-flow detection, as with many JavaScript analysis
  - Scalability problem in whole program analysis

```
domProps {
    innerHTML:...
}
```

Vue, JS-syntax

```
<div v-html=… />
```

Vue, HTML-syntax

Detection Challenges

- **Sink diversity**
  - Many different frameworks, each with their own syntax
  - Multiple syntaxes in the same framework (JavaScript and HTML-like syntax)

```
domProps {
    innerHTML:...
}
```

Vue, JS-syntax

```
<div v-html=… />
```

Vue, HTML-syntax

- **Taint-rule diversity**
  - Dynamic features hinder taint-flow detection, as with many JavaScript analysis
  - Scalability problem in whole program analysis

```
function createPatchFunc(backend) {
  return function patch(vnode) {
    backend.updateDOMProps(vnode)
  }
}
```

Detection Challenges

- **Sink diversity**
  - Many different frameworks, each with their own syntax
  - Multiple syntaxes in the same framework (JavaScript and HTML-like syntax)

```
domProps {
    innerHTML:...
}
```

Vue, JS-syntax

```
<div v-html=… />
```

Vue, HTML-syntax

- **Taint-rule diversity**
  - Dynamic features hinder taint-flow detection, as with many JavaScript analysis
  - Scalability problem in whole program analysis

```
function createPatchFunc(backend) {
  return function patch(vnode) {
    backend.updateDOMProps(vnode)
  }
}
```

How to *comprehensively* and *efficiently* detect taint-style vulnerabilities in SPAs?

Our Approach: Automated Framework Abstraction

- Analyze the SPA runtime, in addition to the SPA component
- From the analysis, abstract the SPA sinks
  - Done once per framework
  - Re-use the result for different application that uses the same framework
- Use the resulting SPA sinks to detect vulnerabilities
  - Using off-the-shelf static analysis tools that allows for taint-rule expansion
  - e.g., CodeQL, Semgrep

TranSPArent Design

Divided into three sub-analyses

Dynamic Autostitch Analysis

- Stitches missing dataflow edge automatically
- Uses information from stack traces

Static Taint Path Analysis

- Analyzes taint path from SPA API to DOM sink
- Abstracts JS-syntax SPA sink

Static Template Mapping Analysis

- Analyzes SPA template engine
- Maps HTML-syntax to JS-syntax SPA sink

TranSPArent Design

Divided into three sub-analyses

Dynamic Autostitch Analysis

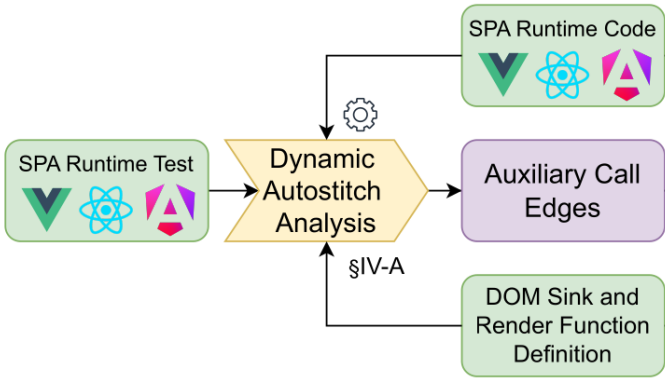
- Stitches missing dataflow edge automatically
- Uses information from stack traces

Static Taint Path Analysis

- Analyzes taint path from SPA API to DOM sink
- Abstracts JS-syntax SPA sink

Static Template Mapping Analysis

- Analyzes SPA template engine
- Maps HTML-syntax to JS-syntax SPA sink



TranSPArent Design

Divided into three sub-analyses

Dynamic Autostitch Analysis

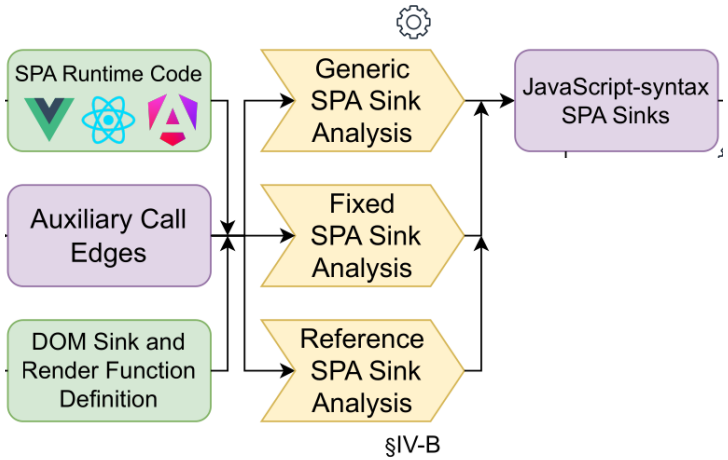
- Stitches missing dataflow edge automatically
- Uses information from stack traces

Static Taint Path Analysis

- Analyzes taint path from SPA API to DOM sink
- Abstracts JS-syntax SPA sink

Static Template Mapping Analysis

- Analyzes SPA template engine
- Maps HTML-syntax to JS-syntax SPA sink



TranSPArent Design

Divided into three sub-analyses

Dynamic Autostitch Analysis

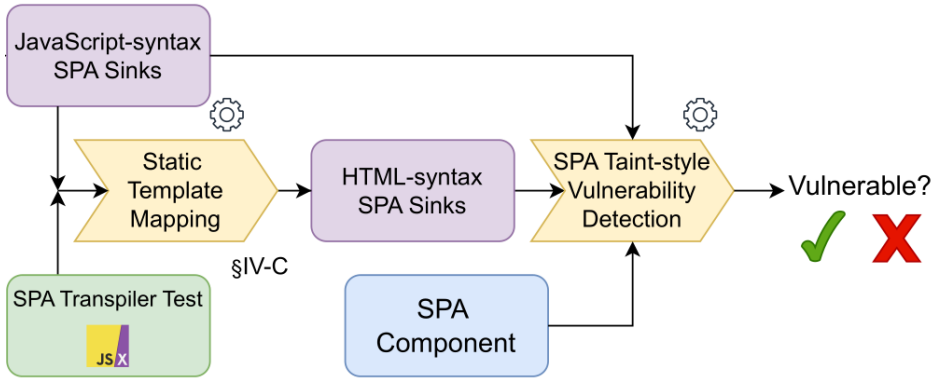
- Stitches missing dataflow edge automatically
- Uses information from stack traces

Static Taint Path Analysis

- Analyzes taint path from SPA API to DOM sink
- Abstracts JS-syntax SPA sink

Static Template Mapping Analysis

- Analyzes SPA template engine
- Maps HTML-syntax to JS-syntax SPA sink



TranSPArent Design

Divided into three sub-analyses

Dynamic Autostitch Analysis

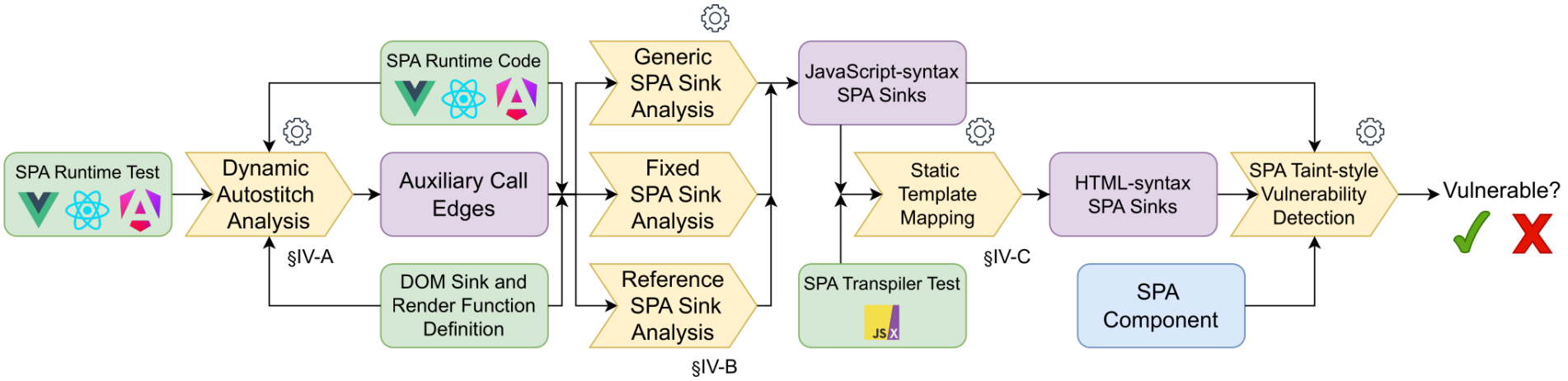
- Stitches missing dataflow edge automatically
- Uses information from stack traces

Static Taint Path Analysis

- Analyzes taint path from SPA API to DOM sink
- Abstracts JS-syntax SPA sink

Static Template Mapping Analysis

- Analyzes SPA template engine
- Maps HTML-syntax to JS-syntax SPA sink

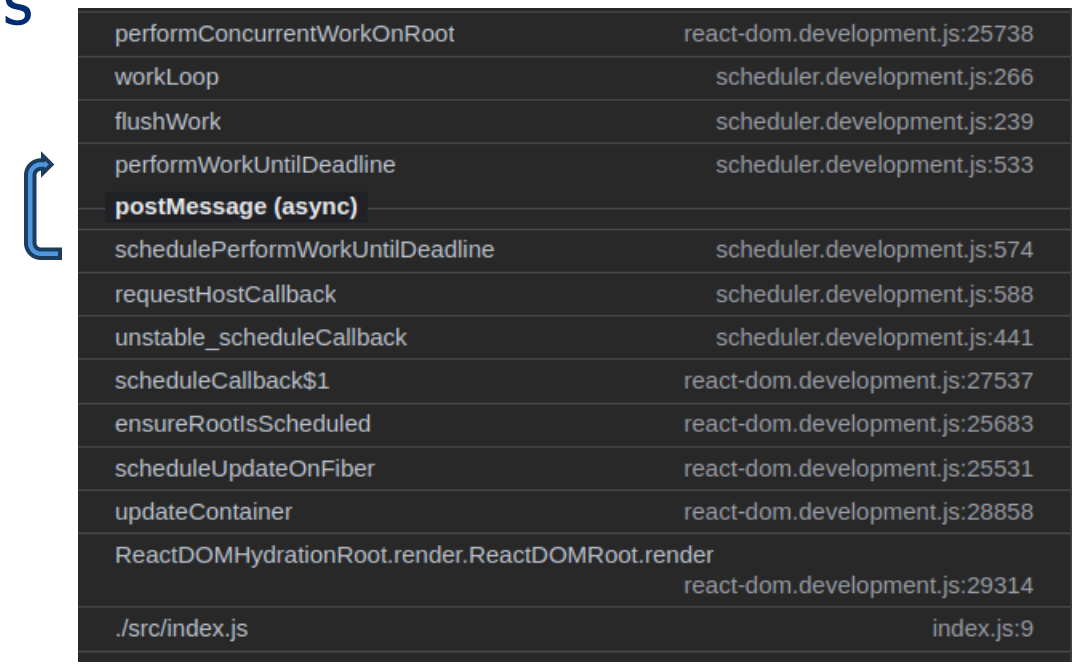


#1: Dynamic Autostitch Analysis

- SPA rendering involves dynamic language features
  - Higher-order function
  - Component-generated function
  - … more in the paper
- Key insight: use unit test suite for missing flows
  - SPA framework has comprehensive test suites
  - Done once per framework, not per application
  - Analyze stack traces that executes a DOM sink
  - Add data flow edges if absent from current graph

#1: Dynamic Autostitch Analysis

- SPA rendering involves dynamic language features
  - Higher-order function
  - Component-generated function
  - … more in the paper
- Key insight: use unit test suite for missing flows
  - SPA framework has comprehensive test suites
  - Done once per framework, not per application
  - Analyze stack traces that executes a DOM sink
  - Add data flow edges if absent from current graph



#2: Static Taint Path Analysis

Automatically abstracts JS-syntax SPA sinks in 3 categories

Generic SPA Sinks

- Expose parameterized sensitive DOM sinks

Fixed SPA Sinks

- Expose sensitive DOM sinks with the right key

Reference SPA Sinks

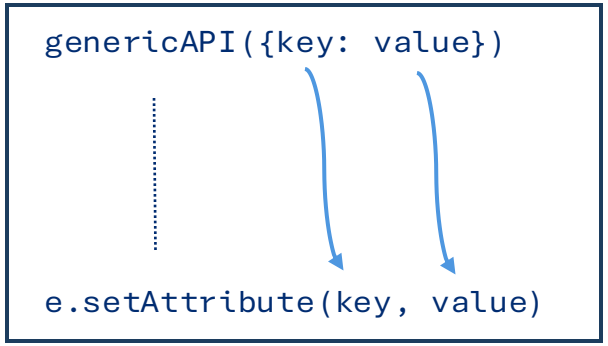
- Expose a raw HTML element that is traditionally enclosed

#2: Static Taint Path Analysis

Automatically abstracts JS-syntax SPA sinks in 3 categories

Generic SPA Sinks

- Expose parameterized sensitive DOM sinks



Fixed SPA Sinks

- Expose sensitive DOM sinks with the right key

Reference SPA Sinks

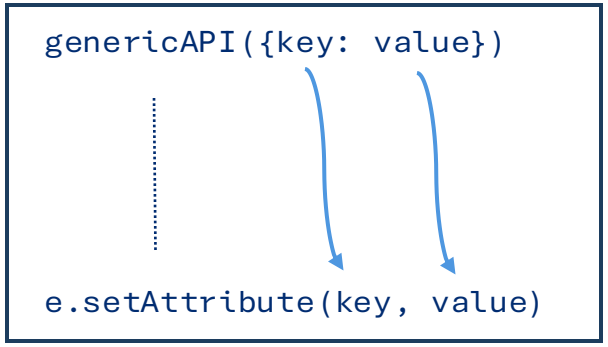
- Expose a raw HTML element that is traditionally enclosed

#2: Static Taint Path Analysis

Automatically abstracts JS-syntax SPA sinks in 3 categories

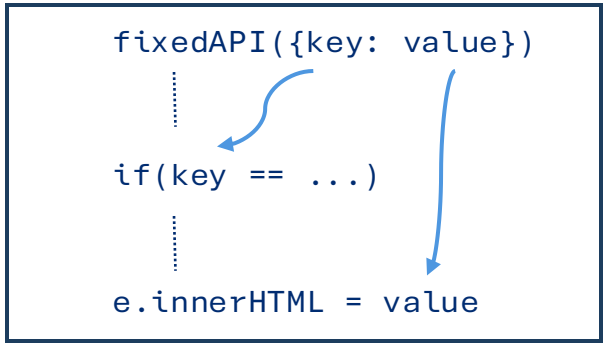
Generic SPA Sinks

- Expose parameterized sensitive DOM sinks



Fixed SPA Sinks

- Expose sensitive DOM sinks with the right key



Reference SPA Sinks

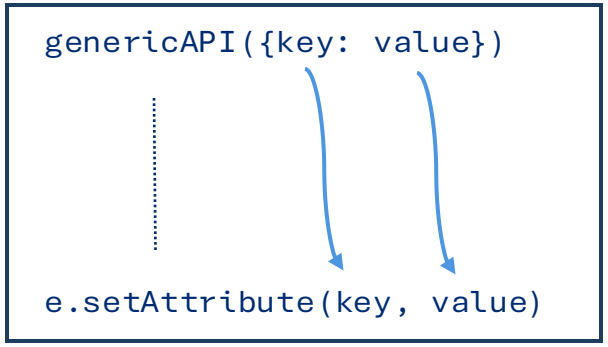
- Expose a raw HTML element that is traditionally enclosed

#2: Static Taint Path Analysis

Automatically abstracts JS-syntax SPA sinks in 3 categories

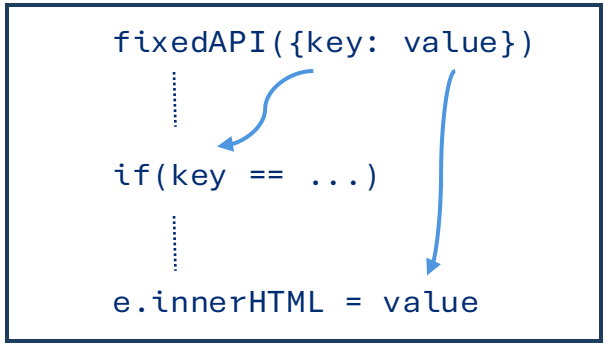
Generic SPA Sinks

- Expose parameterized sensitive DOM sinks



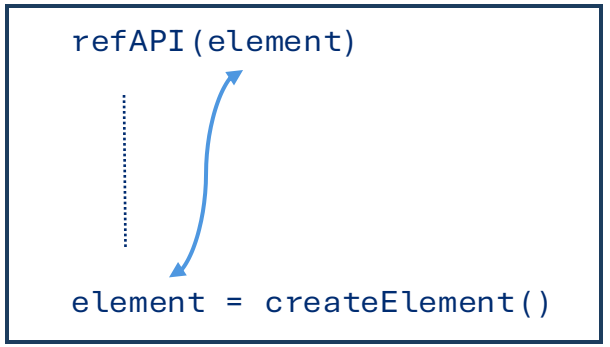
Fixed SPA Sinks

- Expose sensitive DOM sinks with the right key



Reference SPA Sinks

- Expose a raw HTML element that is traditionally enclosed



#3: Static Template Mapping Analysis

- Key insight: every HTML-syntax sink has an equivalent JS-syntax
- Analyze transpiler test suite for HTML-JS syntax mapping
  - Match input and output that has the same value
  - Use JS-syntax SPA sinks from previous analysis
- Two kinds of mapping
  - **Extrapolated mapping**: extrapolate any sensitive attribute / property name
  - **Concrete mapping**: mapping without extrapolation

#3: Static Template Mapping Analysis

- Key insight: every HTML-syntax sink has an equivalent JS-syntax
- Analyze transpiler test suite for HTML-JS syntax mapping
  - Match input and output that has the same value
  - Use JS-syntax SPA sinks from previous analysis
- Two kinds of mapping
  - **Extrapolated mapping**: extrapolate any sensitive attribute / property name
  - **Concrete mapping**: mapping without extrapolation

Test suite

```
transpile("<div domProps-innerHTML='foo'/>")
        value matches ('foo'), compare attribute and property
expect(domProps.innerHTML).toBe('foo')
```

Mapping table

HTML-syntax (attribute)	JS-syntax (properties)
domProps-<nativeProp>	domProps.<nativeProp>
…	…

Evaluation

Zero-day Vulnerability

- TranSPArent found **11** 0-day vulnerabilities
  - Across React, Vue, and Angular
  - Across multiple syntaxes
- Fixed by either adding sanitizers or adding explicit warning in documentation

GitHub Repository Name	Stargazer	Framework	Sinks	Syntax	Vuln. Type	Status
the1812/Bilibili-Evolved	23k+	Vue	domProps.<nativeProp>	JavaScript	XSS	Fixed
apostrophe/apostrophe	4k+	Vue	ref	JavaScript	CSV injection	Reported
bootstrap-vue/bootstrap-vue	14.5k+	Vue	domProps.<nativeProp>	JavaScript	XSS	Reported
alfonsobries/vue-tailwind	2.2k+	Vue	domProps.<nativeProp>	JavaScript	XSS	Reported
miaolz123/vue-markdown	1.9k+	Vue	domProps.<nativeProp>	JavaScript	XSS	Reported
iview/iview	24k+	Vue	domProps.<nativeProp>	JavaScript	XSS	Reported
JosephusPaye/Keen-UI	4.1k+	Vue	domProps.<nativeProp>	JavaScript	XSS	Ack (ext. sanitizer)
jiangshanmeta/vue-admin	170+	Vue	domProps.<nativeProp>	JavaScript	XSS	Reported
surveyjs/survey-library	3.8k+	React	xlinkHref	HTML (JSX)	XSS	Ack (ext. sanitizer)
salesforce/design-system-react	900+	React	xlinkHref	HTML (JSX)	XSS	Acknowledged
evaletolab/ng2-markdown	1+	Angular	ref	JavaScript	XSS	Reported

Accuracy

- Comparison to Vanilla CodeQL (without sink addition)
- Achieves **19.6%** false negative rate among SPA-involved CVEs
  - Expanded library of sinks
  - Main cause: incomplete in-component data flow
- Achieves **42.1%** false discovery rate among GitHub repository dataset
  - Comparable to baseline Vanilla CodeQL
  - Main cause: non-controllable taint source and unidentified sanitization

Tool	FNR	FDR
TranSPArent	11/56 (19.6%)	24/57 (42.1%)
Vanilla CodeQL	35/56 (62.5%)	17/34 (50.0%)

Intermediate SPA Sinks

- Found 19 intermediate SPA sinks
- 14 are not listed by Vanilla CodeQL
- Some frameworks have more sinks than others
  - Supported syntax
  - In-framework sanitizer

Sensitive Framework API	Framework	Syntax	Vanilla CodeQL	ReactApp Scan	Warning	TranSPArent
`attrs.<nativeAttr>`	Vue	JavaScript-syntax	❌	❌	❌	✅
`domProps.<nativeProp>`	Vue	JavaScript-syntax	❌	❌	✅	✅
`ref`	Vue	JavaScript-syntax	❌	❌	❌	✅
`<nativeAttr>`	Vue	HTML-syntax (JSX)	❌	❌	❌	✅
`attrs.<nativeAttr>`	Vue	HTML-syntax (JSX)	❌	❌	❌	✅
`domProps.<nativeProp>`	Vue	HTML-syntax (JSX)	❌	❌	✅	✅
`ref`	Vue	HTML-syntax (JSX)	❌	❌	❌	✅
`<nativeAttr>`	Vue	HTML-syntax (SFC)	⚠️	❌	❌	✅
`v-html`	Vue	HTML-syntax (SFC)	✅	❌	✅	✅
`ref`	Vue	HTML-syntax (SFC)	❌	❌	❌	✅
`<nativeAttr>`	React	JavaScript-syntax	❌	❌	❌	✅
`dangerouslySetInnerHtml`	React	JavaScript-syntax	❌	❌	✅	✅
`ref`	React	JavaScript-syntax	❌	❌	❌	✅
`<nativeAttr>`	React	HTML-syntax (JSX)	⚠️	❌	❌	✅
`dangerouslySetInnerHtml`	React	HTML-syntax (JSX)	✅	✅	✅	✅
`ref`	React	HTML-syntax (JSX)	❌	✅	❌	✅
`renderer2.setProperty`	Angular	JavaScript-syntax	✅	❌	❌	✅
`ref`	Angular	JavaScript-syntax	❌	❌	✅	✅

Conclusion

- SPA vulnerability is prevalent yet understudied
- We designed TranSPArent, an automated tool that abstracts SPA sinks first
  - Generalizable to multiple SPA frameworks
  - Detected real-world vulnerabilities
  - Expanded the capabilities of existing off-the-shelf tools

Thank You for Your Attention!

github.com/diwangs/transparent-ae

Contact
- diwangs@cs.jhu.edu
- @diwangs.dev
- cs.jhu.edu/~diwangs