




 Secrets


 ABAP


 Apex


 C


 C++


 CloudFormation


 COBOL


 C#


 CSS


 Flex


 Go


 HTML


 Java


 **JavaScript**


 Kotlin


 Objective C


 PHP


 PL/I


 PL/SQL


 Python


 RPG


 Ruby


 Scala


 Swift


 Terraform


 Text


 TypeScript

 T-SQL

 VB.NET

 VB6

 XML



JavaScript static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVASCRIPT code

All rules 285

Vulnerability 29

Bug 62


Security Hotspot 43

Code Smell 151


Quick Fix 41


Tags ▾

Search by name... 🔍


 Vulnerability


Cipher algorithms should be robust

 Vulnerability


 Vulnerability


Encryption algorithms should be used with secure mode and padding scheme

 Vulnerability


 Vulnerability


Server hostnames should be verified during SSL/TLS connections

 Vulnerability


 Vulnerability


Server certificates should be verified during SSL/TLS connections

 Vulnerability


 Vulnerability

Cryptographic keys should be robust

 Vulnerability

 Vulnerability

Weak SSL/TLS protocols should not be used

 Vulnerability

 Vulnerability

Origins should be verified during cross-origin communications

 Vulnerability

 Vulnerability

Regular expressions should not be vulnerable to Denial of Service attacks

 Vulnerability

 Vulnerability

File uploads should be restricted

 Vulnerability

 Bug

Function calls should not pass extra arguments

 Bug

 Bug

Regular expressions should be syntactically valid


 Bug


 Bug


Getters and setters should access the

Disabling Vue.js built-in escaping is security-sensitive

Analyze your code

 Security Hotspot

 Blocker

 cwe owasp

Vue.js framework prevents XSS vulnerabilities by automatically escaping HTML contents with the use of native API browsers like `innerText` instead of `innerHTML`.

It's still possible to explicitly use `innerHTML` and similar APIs to render HTML. Accidentally rendering malicious HTML data will introduce an XSS vulnerability in the application and enable a wide range of serious attacks like accessing/modifying sensitive information or impersonating other users.

Ask Yourself Whether

The application needs to render HTML content which:

- could be user-controlled and not previously sanitized.
- is difficult to understand how it was constructed.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

- Avoid injecting HTML content with `v-html` directive unless the content can be considered 100% safe, instead try to rely as much as possible on built-in auto-escaping Vue.js features.
- Take care when using the `v-bind:href` directive to set URLs which can contain malicious Javascript (`javascript:onClick(...)`).
- Event directives like `:onmouseover` are also prone to Javascript injection and should not be used with unsafe values.

Sensitive Code Example

When using Vue.js templates, the `v-html` directive enables HTML rendering without any sanitization:





```
<div v-html="htmlContent"></div> <!-- Noncompliant -->
```

When using a rendering function, the `innerHTML` attribute enables HTML rendering without any sanitization:

```
Vue.component('element', {
  render: function (createElement) {
    return createElement(
      'div',
      {
        domProps: {
          innerHTML: this.htmlContent, // Noncompliant
        }
      }
    );
  },
});
```

https://rules.sonarsource.com/javascript/RSPEC-6299

1/2

Getters and setters should access the expected fields
 Bug
"super()" should be invoked appropriately
 Bug
"Symbol" should not be used as a constructor
 Bug
Results of "in" and "instanceof" should be negated rather than operands
 Bug

When using JSX, the `domPropsInnerHTML` attribute enables HTML rendering without any sanitization:

```
<div domPropsInnerHTML={this.htmlContent}></div> <!-- Noncompliant -->
```

Compliant Solution

When using Vue.js templates, putting the content as a child node of the element is safe:

```
<div>{{ htmlContent }}</div>
```

When using a rendering function, using the `innerText` attribute or putting the content as a child node of the element is safe:

```
Vue.component('element', {
  render: function (createElement) {
    return createElement(
      'div',
      {
        domProps: {
          innerText: this.htmlContent,
        }
      },
      this.htmlContent // Child node
    );
  },
});
```

When using JSX, putting the content as a child node of the element is safe:

```
<div>{this.htmlContent}</div>
```

See

- [OWASP Top 10 2021 Category A3](#) - Injection
- [OWASP Top 10 2017 Category A7](#) - Cross-Site Scripting (XSS)
- [MITRE, CWE-79](#) - Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- [Vue.js - Security - Injecting HTML](#)

Available In:

