

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 rules285

Vulnerability29

Bug62

Security Hotspot43

Code Smell151

Quick Fix41

Tags

Search by name...

Encryption algorithms should be used with secure mode and padding scheme

Vulnerability

Server hostnames should be verified during SSL/TLS connections

Vulnerability

Server certificates should be verified during SSL/TLS connections

Vulnerability

Cryptographic keys should be robust

Vulnerability

Weak SSL/TLS protocols should not be used

Vulnerability

Origins should be verified during cross-origin communications

Vulnerability

Regular expressions should not be vulnerable to Denial of Service attacks

Vulnerability

File uploads should be restricted

Vulnerability

Function calls should not pass extra arguments

Bug

Regular expressions should be syntactically valid

Bug

Getters and setters should access the expected fields

Bug

"super()" should be invoked appropriately

Disabling Angular built-in sanitization is security-sensitive

Analyze your code

Security Hotspot

Blocker

cwe owasp

Angular prevents XSS vulnerabilities by treating all values as untrusted by default. Untrusted values are systematically sanitized by the framework before they are inserted into the DOM.

Still, developers have the ability to manually mark a value as trusted if they are sure that the value is already sanitized. Accidentally trusting malicious 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 value for which sanitization has been disabled is user-controlled.
- It's difficult to understand how this value is constructed.

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

Recommended Secure Coding Practices

- Avoid including dynamic executable code and thus disabling Angular's built-in sanitization unless it's absolutely necessary. Try instead to rely as much as possible on static templates and Angular built-in sanitization to define web page content.
- Make sure to understand how the value to consider as trusted is constructed and never concatenate it with user-controlled data.
- Make sure to choose the correct **DomSanitizer** "bypass" method based on the context. For instance, only use `bypassSecurityTrustUrl` to trust urls in an href attribute context.

Sensitive Code Example

```
import { Component, OnInit } from '@angular/core';
import { DomSanitizer, SafeHtml } from "@angular/platform-br
import { ActivatedRoute } from '@angular/router';





@Component({
  template: '<div id="hello" [innerHTML]="hello"></div>'
})
export class HelloComponent implements OnInit {
  hello: SafeHtml;

  constructor(private sanitizer: DomSanitizer, private route

  ngOnInit(): void {
    let name = this.route.snapshot.queryParams.name;
    let html = "<h1>Hello " + name + "</h1>";
    this.hello = this.sanitizer.bypassSecurityTrustHtml(html
  }
}
```

https://rules.sonarsource.com/javascript/RSPEC-6268

1/2

 Bug
"Symbol" should not be used as a constructor
 Bug
Results of "in" and "instanceof" should be negated rather than operands
 Bug
"in" should not be used with primitive types
 Bug
A compare function should be

Compliant Solution

```
import { Component, OnInit } from '@angular/core';
import { DomSanitizer } from "@angular/platform-browser";
import { ActivatedRoute } from '@angular/router';

@Component({
  template: '<div id="hello"><h1>Hello {{name}}</h1></div>',
})
export class HelloComponent implements OnInit {
  name: string;

  constructor(private sanitizer: DomSanitizer, private route

  ngOnInit(): void {
    this.name = this.route.snapshot.queryParams.name;
  }
}
```

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')
- [Angular - Best Practices - Security](#)

Available In:
 