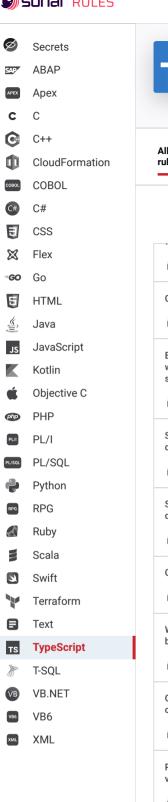
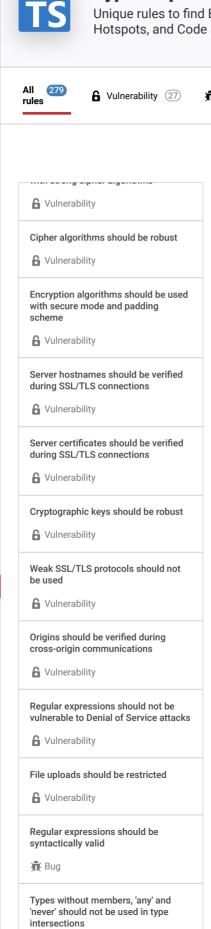
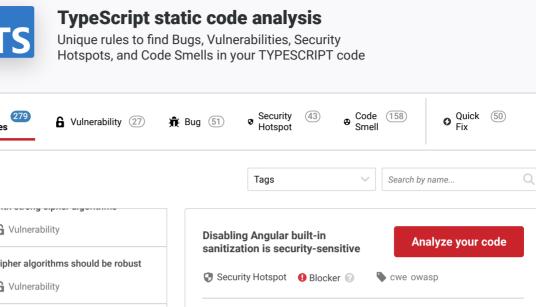


Products ✓







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 <u>DomSanitizer</u> "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 = "<hl>Hello " + name + "</hl>";
    this.hello = this.sanitizer.bypassSecurityTrustHtml(html
}
}
```

₩ Bug

Getters and setters should access the expected fields

R Bug

"super()" should be invoked appropriately

👬 Bug

Results of "in" and "instanceof" should be negated rather than operands

👬 Bug

A compare function should be provided when using "Array.prototype.sort()"

R Bug

## Compliant Solution

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

@Component({
   template: '<div id="hello"><hl>Hello {{name}}</hl></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:

sonarcloud 🐼 | sonarqube

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy