




 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... 🔍

Object literal shorthand syntax should be used

Code Smell

Strings and non-strings should not be added

Code Smell

Object literal syntax should be used

Code Smell

"undefined" should not be assigned

Code Smell

Trailing commas should not be used

Code Smell

Array constructors should not be used

Code Smell

Quotes for string literals should be used consistently

Code Smell

Statements should end with semicolons

Code Smell

Comments should not be located at the end of lines of code

Code Smell

Loops should not contain more than a single "break" or "continue" statement

Code Smell

Variable, property and parameter names should comply with a naming convention

Code Smell

Lines should not end with trailing whitespaces

Code Smell

Reading the Standard Input is security-sensitive

Analyze your code

Security Hotspot

Critical

Reading Standard Input is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2005-2337
- CVE-2017-11449

It is common for attackers to craft inputs enabling them to exploit software vulnerabilities. Thus any data read from the standard input (stdin) can be dangerous and should be validated.

This rule flags code that reads from the standard input.

Ask Yourself Whether

- data read from the standard input is not sanitized before being used.

You are at risk if you answered yes to this question.

Recommended Secure Coding Practices

Sanitize all data read from the standard input before using it.

Sensitive Code Example

```
// The process object is a global that provides information
// All uses of process.stdin are security-sensitive and should be sanitized

process.stdin.on('readable', () => {
  const chunk = process.stdin.read(); // Sensitive
  if (chunk !== null) {
    dosomething(chunk);
  }
});

const readline = require('readline');
readline.createInterface({
  input: process.stdin // Sensitive
}).on('line', (input) => {
  dosomething(input);
});
```

See

- MITRE, CWE-20 - Improper Input Validation







Deprecated

This rule is deprecated, and will eventually be removed.

Available In:

https://rules.sonarsource.com/javascript/RSPEC-4829

1/2

<div>Files should contain an empty newlne at the end</div> <div> Code Smell</div>	<div>sonarcloud  sonarqube </div> <div><div>© 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</div></div>
<div>An open curly brace should be located at the end of a line</div> <div> Code Smell</div>	
<div>Tabulation characters should not be used</div> <div> Code Smell</div>	
<div>Function and method names should comply with a naming convention</div> <div> Code Smell</div>	