




 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

Using regular expressions is security-sensitive

Analyze your code

Security HotspotCritical

Using regular expressions is security-sensitive. It has led in the past to the following vulnerabilities:

- CVE-2017-16021
- CVE-2018-13863

Evaluating regular expressions against input strings is potentially an extremely CPU-intensive task. Specially crafted regular expressions such as (a+)+s will take several seconds to evaluate the input string aaaaaaaaaaaaaaaaaaaaaaaaaaaaaabs. The problem is that with every additional a character added to the input, the time required to evaluate the regex doubles. However, the equivalent regular expression, a+s (without grouping) is efficiently evaluated in milliseconds and scales linearly with the input size.

Evaluating such regular expressions opens the door to [Regular expression Denial of Service \(ReDoS\)](#) attacks. In the context of a web application, attackers can force the web server to spend all of its resources evaluating regular expressions thereby making the service inaccessible to genuine users.

This rule flags any execution of a hardcoded regular expression which has at least 3 characters and at least two instances of any of the following characters: \*+{.

Example: (a+)\*

Ask Yourself Whether

- the executed regular expression is sensitive and a user can provide a string which will be analyzed by this regular expression.
- your regular expression engine performance decrease with specially crafted inputs and regular expressions.

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

Recommended Secure Coding Practices

Check whether your regular expression engine (the algorithm executing your regular expression) has any known vulnerabilities. Search for vulnerability reports mentioning the one engine you're using.

Use if possible a library which is not vulnerable to Redos Attacks such as [Google Re2](#).

Remember also that a ReDos attack is possible if a user-provided regular expression is executed. This rule won't detect this kind of injection.





Sensitive Code Example

```
const regex = /(a+)+b/; // Sensitive
const regex2 = new RegExp("(a+)+b"); // Sensitive

str.search("(a+)+b"); // Sensitive
str.match("(a+)+b"); // Sensitive
str.split("(a+)+b"); // Sensitive
```

https://rules.sonarsource.com/javascript/RSPEC-4784

1/2

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

Note: String.matchAll does not raise any issue as it is not supported by NodeJS.

Exceptions

Some corner-case regular expressions will not raise an issue even though they might be vulnerable. For example: ( a | aa ) + , ( a | a ? ) + .

It is a good idea to test your regular expression if it has the same pattern on both side of a " | " .

See

- [OWASP Top 10 2017 Category A1](#) - Injection
- [MITRE, CWE-624](#) - Executable Regular Expression Error
- OWASP Regular expression Denial of Service - ReDoS

Deprecated

This rule is deprecated; use {rule:javascript:S5852} instead.

Available In:

