sonar RULES Products

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...

Using Sockets is security-sensitive
Security Hotspot

Executing XPath expressions is security-sensitive
Security Hotspot

Encrypting data is security-sensitive
Security Hotspot

Using regular expressions is security-sensitive
Security Hotspot

Class methods should be used instead of "prototype" assignments
Code Smell

Function constructors should not be used
Code Smell

Variables should be declared with "let" or "const"
Code Smell

Unchanged variables should be marked "const"
Code Smell

Wildcard imports should not be used
Code Smell

"switch" statements should not be nested
Code Smell

Cyclomatic Complexity of functions should not be too high
Code Smell

"strict" mode should be used with caution
Code Smell

Tests should check which exception is thrown

Analyze your code

Code Smell Major tests chai mocha

It is not good enough to test if an exception is raised, without checking which exception it is. Such tests will not be able to differentiate the expected exception from an unexpected one. They should instead validate the exception message and/or type.

This rule raises an issue in the following cases:

- When an asynchronous Mocha test calls the `done()` callback, without parameters, in a `catch` block and there is no reference to the caught exception in this block. Either the error should be passed to `done()` or the exception should be checked further.
- When Chai assertions are used to test if a function throws any exception, or an exception of type `Error` without checking the message.
- When Chai assertions are used to test if a function does not throw an exception of type `Error` without checking the message.

Rule doesn't raise an issue when assertion is negated using `not`, it such case the exception doesn't need to be specific.

Noncompliant Code Example

```
const expect = require("chai").expect;
const fs = require("fs");

describe("exceptions are not tested properly", function() {
    const funcThrows = function () { throw new TypeError('Wh
    const funcNoThrow = function () { /*noop*/ };

    it("forgot to pass the error to 'done()'", function(done
        fs.readFile("/etc/zshrc", 'utf8', function(err, data
            try {
                expect(data).to.match(/some expected string/
            } catch (e) {  // Noncompliant
                // Either the exception should be passed to
                done();
            }
        });
    });

    it("does not 'expect' a specific exception", function()
        expect(funcThrows).to.throw();  // Noncompliant
        // Error is not precise enough
        expect(funcThrows).to.throw(Error);  // Noncompliant
    });
});
```

Compliant Solution

```
const expect = require("chai").expect;
const { AssertionError } = require('chai');
```

Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply

Code Smell

"switch" statements should have "default" clauses

Code Smell

"if ... else if" constructs should end with "else" clauses

Code Smell

Control structures should use curly braces

Code Smell

```
const fs = require("fs");

describe("exceptions are tested properly", function() {
    const funcThrows = function () { throw new TypeError('Wh
    const funcNoThrow = function () { /*noop*/ };

    it("forgot to pass the error to 'done()'", function(done
        fs.readFile("/etc/zshrc", 'utf8', function(err, data
            try {
                expect(data).to.match(/some expected string/
            } catch (e) {
                expect(e).to.be.an.instanceof(AssertionError
                done();
            }
        });
    });

    it("does not 'expect' a specific exception", function()
        expect(funcThrows).to.throw(TypeError);
        expect(funcNoThrow).to.not.throw(Error, /My error me
    });
});
```

Available In:

sonarlint | sonarcloud | sonarqube