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...

used
Vulnerability

Variables should be defined before being used
Bug

Variables declared with "var" should be declared before they are used
Code Smell

Track lack of copyright and license headers
Code Smell

Reading the Standard Input is security-sensitive
Security Hotspot

Using command line arguments is security-sensitive
Security Hotspot

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

Chai assertions should have only one reason to succeed

Analyze your code

Code Smell | Major | chai tests

Each assertion should test one condition and have only one reason to fail or succeed. If an assertion success depends on multiple conditions it becomes difficult to understand if the test passed for the right reason. It also makes debugging more difficult when the test fails.

This rule raises an issue when the following Chai assertions are found:

- When `.not` and `.throw` are used together and at least one argument is provided to `.throw`. Such assertions succeed when the target either does not throw any exception, or when it throws an exception different from the one provided.
- When `.not` and `.include` are used together and an `object` is given to `.include`. Such assertions succeed when one or multiple key/values are missing.
- When `.not` and `.property` are used together and `.property` is given at least two arguments. Such assertions succeed when the target either doesn't have the requested property, or when this property exists but has a different value.
- When `.not` and `.ownPropertyDescriptor` are used together and `.ownPropertyDescriptor` is given at least two arguments. Such assertions succeed when the target either doesn't have the requested property descriptor, or its property descriptor is not deeply equal to the given descriptor
- When `.not` and `.members` are used together. Such assertions succeed when one or multiple members are missing.
- When `.change` and `.by` are used together. Such assertions succeed when the target either decreases or increases by the given delta
- When `.not` and `.increase` are used together. Such assertions succeed when the target either decreases or stays the same.
- When `.not` and `.decrease` are used together. Such assertions succeed when the target either increases or stays the same.
- When `.not` negates `.by`. Such assertions succeed when the target didn't change by one specific delta among all the possible deltas.
- When `.not` and `.finite` are used together. Such assertions succeed when the target either is not a `number`, or is one of `Nan`, positive `Infinity`, negative `Infinity`.

Noncompliant Code Example

```
const expect = require('chai').expect;

describe("uncertain assertions", function() {
    const throwsTypeError = () => { throw new TypeError() }

    it("uses chai 'expect'", function() {
        expect(throwsTypeError).to.not.throw(ReferenceError)
        expect({a: 42}).to.not.include({b: 10, c: 20});  //
        expect({a: 21}).to.not.have.property('b', 42); // No
        expect({a: 21}).to.not.have.ownPropertyDescriptor('b
            configurable: true,
            enumerable: true,
            writable: true,
```

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

```
            value: 42,
        });
        expect([21, 42]).to.not.have.members([1, 2]); // Non

        var myObj = { value: 1 }
        const incThree = () => { myObj.value += 3; };
        const decThree = () => { myObj.value -= 3; };
        const doNothing = () => {};

        expect(incThree).to.change(myObj, 'value').by(3); //
        expect(decThree).to.change(myObj, 'value').by(3); //

        expect(decThree).to.not.increase(myObj, 'value'); //
        expect(incThree).to.not.decrease(myObj, 'value'); //

        expect(doNothing).to.not.increase(myObj, 'value'); /
        expect(doNothing).to.not.decrease(myObj, 'value'); /

        expect(incThree).to.increase(myObj, 'value').but.not

        let toCheck;
        expect(toCheck).to.not.be.finite; // Noncompliant
    });
});
```

Compliant Solution

```
const expect = require('chai').expect;

describe("uncertain assertions", function() {
    const throwsTypeError = () => { throw new TypeError() }

    it("uses chai 'expect'", function() {
        expect(throwsTypeError).to.throw(TypeError)
        expect({a: 42}).to.not.have.any.keys('b', 'c');
        expect({a: 21}).to.not.have.property('b');
        expect({a: 21}).to.not.have.ownPropertyDescriptor('b
        expect([21, 42]).to.not.include(1).and.not.include(2

        var myObj = { value: 1 }
        const incThree = () => { myObj.value += 3; };
        const decThree = () => { myObj.value -= 3; };
        const doNothing = () => {};

        expect(incThree).to.increase(myObj, 'value').by(3);
        expect(decThree).to.decrease(myObj, 'value').by(3);

        expect(decThree).to.decrease(myObj, 'value').by(3);
        expect(incThree).to.increase(myObj, 'value').by(3);

        expect(doNothing).to.not.change(myObj, 'value');

        expect(incThree).to.increase(myObj, 'value').by(3);

        let toCheck;
        // Either of the following is valid
        expect(toCheck).to.be.a('string');
        expect(toCheck).to.be.NaN;
        expect(toCheck).to.equal(Infinity);
        expect(toCheck).to.equal(-Infinity);
    });
});
```

Available In:

sonarlint | sonarcloud | sonarqube