JavaScript static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your JAVASCRIPT code

modify the loops' counters
Code Smell

Functions should not be empty
Code Smell

Server-side requests should not be vulnerable to forging attacks
Vulnerability

Non-empty statements should change control flow or have at least one side-effect
Bug

Regular expressions with the global flag should be used with caution
Bug

Replacement strings should reference existing regular expression groups
Bug

Regular expressions should not contain control characters
Bug

Alternation in regular expressions should not contain empty alternatives
Bug

Mocha timeout should be disabled by setting it to "0".
Bug

Unicode Grapheme Clusters should be avoided inside regex character classes
Bug

Assertions should not be given twice the same argument
Bug

Alternatives in regular expressions should be grouped when used with

Regular expressions should not be vulnerable to Denial of Service attacks

Analyze your code

Vulnerability Critical injection cwe owasp denial-of-service

Most of the regular expression engines use `backtracking` to try all possible execution paths of the regular expression when evaluating an input, in some cases it can cause performance issues, called `catastrophic backtracking` situations. In the worst case, the complexity of the regular expression is exponential in the size of the input, this means that a small carefully-crafted input (like 20 chars) can trigger `catastrophic backtracking` and cause a denial of service of the application. Super-linear regex complexity can lead to the same impact too with, in this case, a large carefully-crafted input (thousands chars).

It is not recommended to construct a regular expression pattern from a user-controlled input, if no other choice, sanitize the input to remove/annihilate regex metacharacters.

Noncompliant Code Example

Example with RegExp:

```
function (req, res) {
  const pattern = RegExp(req.query.pattern); // Noncompliant
  pattern.test(req.query.input);
};
```

Don't use safe-regex kind of libraries to validate regexes, as it is prone to FPs/FNs:

```
const safe = require('safe-regex');

function (req, res) {
  // Not a good validator, eg of FN: http://test/noncomplian
  if(safe(req.query.pattern)) {
    const regex = RegExp(req.query.pattern); // Noncompliant
    regex.test(req.query.input);
  }
};
```

Compliant Solution

Escape regex special characters:

```
const escapeStringRegexp = require('escape-string-regexp');

function (req, res) {
  const pattern = RegExp(escapeStringRegexp(req.query.patter
  pattern.test(req.query.input);
};
```

See

anchors

Bug

Promise rejections should not be caught by 'try' block

Bug

Collection elements should not be replaced unconditionally

Bug

Errors should not be created without being thrown

Bug

Collection sizes and array length

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-400 - Uncontrolled Resource Consumption
- MITRE, CWE-1333 - Inefficient Regular Expression Complexity
- OWASP Regular expression Denial of Service - ReDoS

Available In:

sonarcloud sonarqube Developer Edition