JavaScript static code analysis

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

Dynamic code execution should not be vulnerable to injection attacks

Analyze your code

Vulnerability Blocker injection cwe owasp sans-top25

Applications that execute code dynamically should neutralize any externally-provided values used to construct the code. Failure to do so could allow an attacker to execute arbitrary code. This could enable a wide range of serious attacks like accessing/modifying sensitive information or gain full system access.

The mitigation strategy should be based on whitelisting of allowed values or casting to safe types.

Noncompliant Code Example

`eval` and other functions that dynamically execute code should not be used in combination with user-input:

```
let input = req.query.input;
eval(input); // Noncompliant
(Function(input))(); // Noncompliant
(new Function(input))(); // Noncompliant
```

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

let input = req.query.input;
vm.runInThisContext(input);  // Noncompliant
const context = {};
vm.createContext(context);
vm.runInContext(input, context); // Noncompliant
vm.runInNewContext(input, context); // Noncompliant
vm.compileFunction(input)(); // Noncompliant
(new vm.Script(input)).runInThisContext(); // Noncompliant
```

```
var Module = require('module');

let name = req.query.name;
let input = req.query.input;
var mod = new Module(name, module.parent);
mod._compile(input, name); // Noncompliant
```

In a MongoDB context, arbitrary Javascript code can be executed with the `$where` operator for instance:

```
let username = req.query.username;
query = { $where: `this.username == '${username}'` } // Nonc
User.find(query, function (err, users) {
  if (err) {
    // Handle errors
  } else {
    res.render('userlookup', { title: 'User Lookup', users:
```

Tests should include assertions

Code Smell

"future reserved words" should not be used as identifiers

Code Smell

Octal values should not be used

Code Smell

Switch cases should end with an unconditional "break" statement

Code Smell

"switch" statements should not

```
  }
});
```

Compliant Solution

If `eval` is used to parse a JSON string this should be done instead with `JSON.parse`:

```
let obj = JSON.parse(req.query.input); // Compliant
```

In a MongoDB context, don't use `$where` operator or validate the data:

```
let username = req.query.username;
query = { username: username } // Compliant
User.find(query, function (err, users) {
  if (err) {
    // Handle errors
  } else {
    res.render('userlookup', { title: 'User Lookup', users:
  }
});
```

See

- OWASP Top 10 2021 Category A3 - Injection
- OWASP Top 10 2017 Category A1 - Injection
- MITRE, CWE-20 - Improper Input Validation
- MITRE, CWE-95 - Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
- SANS Top 25 - Risky Resource Management

Available In:

sonarcloud sonarqube Developer Edition