

3 CSS

 \mathbb{X} Flex

-GO

Go 5 HTML

Java

JavaScript

Kotlin

Kubernetes

Objective C

PHP

PL/I

PL/SQL

Python

RPG

Ruby

Scala

Swift

Terraform

Text

TypeScript

T-SQL

VB.NET

VB6

XML



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

⊗ Code ΑII 311 Security 18 206 6 Vulnerability (13) ₩ Bug (74) rules Hotspot Smell

"memset" should not be used to delete Functions should be declared sensitive data

6 Vulnerability

POSIX functions should not be called with arguments that trigger buffer overflows

♠ Vulnerability

XML parsers should not be vulnerable to XXE attacks

Vulnerability

Function-like macros should not be invoked without all of their arguments

₩ Bug

The address of an automatic object should not be assigned to another object that may persist after the first object has ceased to exist

👬 Bug

"pthread_mutex_t" should be unlocked in the reverse order they were locked

Bug

"pthread_mutex_t" should be properly initialized and destroyed

Bua

"pthread_mutex_t" should not be consecutively locked or unlocked

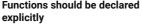
Bug

Functions with "noreturn" attribute should not return

₩ Bua

"memcmp" should only be called with pointers to trivially copyable types with no padding

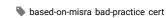
🖷 Bug



Tags

Analyze your code

O Quick 14



Search by name.

The use of prototypes enables the compiler to check the integrity of function definitions and calls. Without prototypes the compiler is not obliged to pick up certain errors in function calls (e.g. different number of arguments from the function body, mismatch in types of arguments between call and definition). Function interfaces have been shown to be a cause of considerable problems, and therefore this rule is considered very important.

The recommended method of implementing function prototypes for external functions is to declare the function (i.e. give the function prototype) in a header file, and then include the header file in all those code files that need the prototype (see MISRA C 2004, Rule 8.8).

Noncompliant Code Example

```
void example() {
 fun(); // Noncompliant
void fun() {
```

Compliant Solution

```
void fun();
void example() {
  fun();
void fun() {
```

See

- MISRA C:2004, 8.1 Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call
- MISRA C:2012, 17.3 A function shall not be declared implicitly
- CERT, DCL07-C. Include the appropriate type information in function declarators
- CERT, DCL31-C. Declare identifiers before using them

• MISRA C:2004, 8.8 - An external object or function shall be declared in one and only one file

Available In:

sonarlint ⊕ | sonarcloud 💩 | sonarqube

Developer

Stack allocated memory and nonowned memory should not be freed

🕕 Bug

Closed resources should not be accessed

👬 Bug

Dynamically allocated memory should be released

👬 Bug

Freed memory should not be used

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy