Secrets
ABAP
Apex
C
C++
CloudFormation
COBOL
C#
CSS
Flex
Go
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

C static code analysis

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

All rules 311 | Vulnerability 13 | Bug 74 | Security Hotspot 18 | Code Smell 206 | Quick Fix 14

Tags

Search by name...

"memset" should not be used to delete sensitive data
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
Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice
Bug

Functions with "noreturn" attribute should not return
Bug

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

Function-like macros should not be used

Analyze your code

Code Smell | Critical | cppcoreguidelines based-on-misra preprocessor bad-practice cert

It is tempting to treat function-like macros as functions, but the two things work differently. For instance, the use of functions offers parameter type-checking, while the use of macros does not. Additionally, with macros, there is the potential for a macro to be evaluated multiple times. In general, functions offer a safer, more robust mechanism than function-like macros, and that safety usually outweighs the speed advantages offered by macros. Therefore functions should be used instead when possible.

Noncompliant Code Example

```
#define CUBE (X) ((X) * (X) * (X)) // Noncompliant

void func(void) {
  int i = 2;
  int a = CUBE(++i); // Noncompliant. Expands to: int a = ((+
  // ...
}
```

Compliant Solution

```
inline int cube(int i) {
  return i * i * i;
}

void func(void) {
  int i = 2;
  int a = cube(++i); // yields 27
  // ...
}
```

See

- MISRA C:2004, 19.7 - A function should be used in preference to a function-like macro.
- MISRA C++:2008, 16-0-4 - Function-like macros shall not be defined.
- MISRA C:2012, Dir. 4.9 - A function should be used in preference to a function-like macro where they are interchangeable
- CERT, PRE00-C. - Prefer inline or static functions to function-like macros
- C++ Core Guidelines ES.31 - Don't use macros for constants or "functions"

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition

Stack allocated memory and non-owned 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