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

The right-hand operands of && and || should not contain side effects

Analyze your code

Code Smell | Blocker | based-on-misra cert

There are some situations in C++ where certain parts of expressions may not be evaluated. If these sub-expressions contain side effects then those side effects may or may not occur, depending on the values of other sub expressions. The operators which can lead to this problem are `&&` and `||`, where the evaluation of the right-hand operand is conditional on the value of the left-hand operand. The conditional evaluation of the right-hand operand of one of the logical operators can easily cause problems if the developer relies on a side effect occurring.

Operations that cause side effects are:

- accessing a volatile object
- modifying an object
- modifying a file
- calling a function that performs any operations that cause changes in the state of the execution environment of the calling function.

This rule raises an issue when there is assignment or the use of the increment/decrement operators in right-hand operands.

Noncompliant Code Example

```
if ( ishigh && ( x == i++ ) ) // Noncompliant
...
if ( ishigh && ( x ==  getX() ) ) // Only acceptable if getX(
```

The operations that cause side effects are accessing a volatile object, modifying an object, modifying a file, or calling a function

that does any of those operations, which cause changes in the state of the execution environment of the calling function.

For the time being, this rule only check that there is no assignment or no use of increment/decrement operators made in right hand operands.

See

- MISRA C:2004, 12.4 - The right-hand operand of a logical && or || operator shall not contain side effects.
- MISRA C++:2008, 5-14-1 - The right hand operand of a logical && or || operator shall not contain side effects.
- MISRA C:2012, 13.5 - The right hand operand of a logical && or || operator shall not contain persistent side effects
- CERT, EXP02-C. - Be aware of the short-circuit behavior of the logical AND and OR operators

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