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

## "bool" expressions should not be used as operands to built-in operators other than =, &&, ||, !, ==, !=, unary &, and the conditional operator

Analyze your code

Code Smell | Major | based-on-misra suspicious

The use of `bool` operands with other operators is unlikely to be meaningful (or intended). Best case it will be confusing to maintainers, worst case it will not have the intended effect. Either way, it is highly recommended to stick to boolean operators when dealing with `bool` operands.

This rule allows the detection of such uses, which often occur because the logical operators (`&&`, `||` and `!`) can be easily confused with the bitwise operators (`&`, `|` and `~`).

### Noncompliant Code Example

```
bool b1 = true;
bool b2 = false;
int8_t s8a;
if ( b1 & b2 ) // Noncompliant
if ( ~b1 ) // Noncompliant
if ( b1 < b2 ) // Noncompliant
if ( b1 ^ b2 ) // Noncompliant
```

### Compliant Solution

```
if ( b1 && b2 )
if ( !b1 )
if ( b1 == false )
if ( b1 == b2 )
if ( b1 != b2 )
s8a = b1 ? 3 : 7;
```

### Exceptions

Operators `|=` and `&=` are ignored when used with `bool` operands. Operator `++` is also ignored with a `bool` operand because it is covered by rule {rule:cpp:S2668}.

```
void test(bool b1, bool b2, int i1) {
  b1 |= b2; // ignored
  b1++; // ignored here, handled by S2668
  b1 &= b2; // ignored
  b1 &= i1; // Noncompliant; right operand is not a bool
}
```

### See

- MISRA C++:2008, 4-5-1 - Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, ||, !, the equality operators == and !=, the unary & operator, and the conditional operator.

**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**

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition