



**ABAP** 

<sub>АРЕХ</sub> Арех

**c** c

C++

CloudFormation

COBOL COBOL

C# C#

**E** CSS

**⊠** Flex

=GO

5 HTML

Go

🐇 Java

Js JavaScript

Kotlin

Kubernetes

6 Objective C

PHP

PL/I

PL/SQL PL/SQL

Python

RPG RPG

Ruby

Scala

Swift

**Terraform** 

**Text** 

TS TypeScript

T-SQL

VB VB.NET

VB6 VB6

XML XML



# C++ static code analysis

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

All 578 rules Vulnerability 13

**R** Bug 111

R Bug

Security Hotspot

Och Code (436)

Quick 68 Fix

Tags

Search by name...

🔷 cppcoreguidelines based-on-misra cert

"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

Assigning to an optional should directly target the optional

🛊 Bug

Result of the standard remove algorithms should not be ignored

🕀 Bug

"std::scoped\_lock" should be created with constructor arguments

<table-of-contents> Bug

Objects should not be sliced

📆 Bug

Immediately dangling references should not be created

📆 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

Boolean operations should not have numeric operands, and vice versa

Analyze your code

There are several constructs in the language that work with boolean:

• If statements: if (b) ...

• Conditional operator: int i = b ? 0 : 42;

• Logical operators: (b1 || b2) && !b3

Those operations would also work with arithmetic or enum values operands, because there is a conversion from those types to bool. However, this conversion might not always be obvious, for instance, an integer return code might use the value 0 to indicate that everything worked as expected, but converted to boolean, this value would be false, which often denotes failure. Conversion from integer to bool should be explicit.

Moreover, a logical operation with integer types might also be a confusion with the bitwise operators ( $\alpha$ , | and  $\sim$ ).

Converting a pointer to bool to check if it is null is idiomatic and is allowed by this rule. We also allow the use of any user-defined type convertible to bool (for instance std::ostream), since they were specifically designed to be used in such situations. What this rule really detects is the use or arithmetic types (int, long...) and of enum types.

On the other hand, arithmetic operations are defined with booleans, but usually make little sense (think of adding two booleans). Booleans should not be used in an arithmetic context.

Finally, comparing a boolean with the literals true or false is unnecessarily verbose, and should be avoided.

## Noncompliant Code Example

```
if ( 1 && ( c < d ) ) // Noncompliant
if ( ( a < b ) && ( c + d ) ) // Noncompliant
if ( u8_a && ( c + d ) ) // Noncompliant
if ( !0 ) // Noncompliant, always true
if ( !ptr ) // Compliant
if ( ( a < b ) && ( c < d ) ) // Compliant
if ( !false ) // Compliant
if ( !false ) // Compliant
if ( !!a) // Compliant by exception
if ( ( a < b ) == true) // Noncompliant</pre>
```

#### **Compliant Solution**

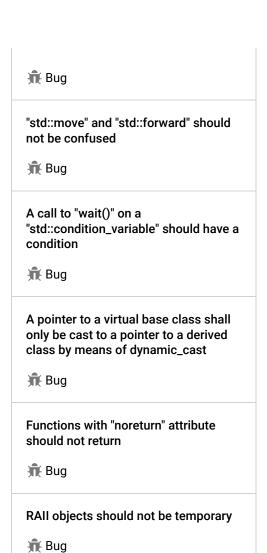
```
if ( 1 != 0 \&\& (c < d)) // Compliant, but left operand is if ( (a < b) \&\& (c + d) != 0) // Compliant if ( u8\_a != 0 \&\& (c + d) != 0) // Compliant if ( 0 == 0) // Compliant, always true if ( a < b)
```

### Exceptions

Some people use !! as a shortcut to cast an integer to bool. This usage of the ! operator with an integer argument is allowed for this rule.

#### See

• MISRA C:2004, 12.6 - The operands of logical operators (&&, || and !) should be



"memcmp" should only be called with pointers to trivially copyable types

"memcpy", "memmove", and "memset" should only be called with pointers to

"std::auto\_ptr" should not be used

Destructors should be "noexcept"

with no padding

trivially copyable types

📆 Bug

Rug Bug

🕀 Bug

📆 Bug

effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&,  $\parallel$  and !).

- MISRA C++:2008, 5-3-1 Each operand of the ! operator, the logical && or the logical || operators shall have type bool.
- CERT, EXP54-J. Understand the differences between bitwise and logical operators
- CERT, EXP13-C. Treat relational and equality operators as if they were nonassociative
- C++ Core Guidelines ES.87 Don't add redundant == or != to conditions

Available In:

sonarlint 😊 | sonarcloud 💩 | sonarqube | Developer Edition

 $@\ 2008-2022\ Sonar Source\ S.A.,\ Switzerland.\ All\ content\ is\ copyright\ protected.\ SONAR,$ 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