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

Objective C static code analysis

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

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

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

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

Boolean operations should not have numeric operands, and vice versa

Analyze your code

Bug Major cppcoreguidelines based-on-misra cert

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 (`&`, `|` and `~`).

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 (!!a) // Compliant by exception
if ( ( a < b ) == true) // Noncompliant
```

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

Freed memory should not be used

Bug

Memory locations should not be released more than once

Bug

Memory access should be explicitly bounded to prevent buffer overflows

Bug

Printf-style format strings should not lead to unexpected behavior at runtime

Bug

Recursion should not be infinite

Bug

Resources should be closed

Bug

Hard-coded credentials are security-sensitive

Security Hotspot

"goto" should jump to labels declared later in the same function

Code Smell

Only standard forms of the "defined" directive should be used

Code Smell

Switch labels should not be nested inside non-switch blocks

Code Smell

effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&, || 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:
sonarcloud sonarqube Developer Edition