C static code analysis

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

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

Identical expressions should not be used on both sides of a binary operator

Analyze your code

Bug Major cert

Using the same value on either side of a binary operator is almost always a mistake. In the case of logical operators, it is either a copy/paste error and therefore a bug, or it is simply wasted code, and should be simplified. In the case of bitwise operators and most binary mathematical operators, having the same value on both sides of an operator yields predictable results, and should be simplified.

Noncompliant Code Example

```
if ( a == a ) { // always true
  do_z();
}
if ( a != a ) { // always false
  do_y();
}
if ( a == b && a == b ) { // if the first one is true, the se
  do_x();
}
if (a == b || a == b ) { // if the first one is true, the sec
  do_w();
}

if (5 / 5) { // always 1
  do_v();
}
if (5 - 5) { // always 0
  do_u();
}
```

Exceptions

The following are ignored:

- The expression `1 << 1`
- When an increment or decrement operator is used, ex: `*p++ == *p++`
- Bitwise operators `|, &, ^`
- Arithmetic operators `+, *`
- Assignment operators `=, +=, *=`

See

- CERT, MSC12-C. - Detect and remove code that has no effect or is never executed
- {rule:cpp:S1656} - Implements a check on =.

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