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

"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

Signed and unsigned types should not be mixed in expressions

Analyze your code

Code Smell Minor based-on-misra cert

Some signed to unsigned conversions may lead to implementation-defined behavior. This behavior may not be consistent with developer expectations.

If you need to mix signed and unsigned types, you should make your intent explicit by using explicit casts and avoiding implicit casts.

This rule will detect implicit conversions that change the signedness.

Noncompliant Code Example

```
void f(int a) {
  unsigned int b = a; // Noncompliant
  int c = (a > 0) ? a : b; // Noncompliant

  if (a > b) { // Noncompliant
    // ...
  }
}
```

Compliant Solution

```
void f(int a) {
  unsigned int b = static_cast<unsigned int>(a); // Compliant
}
```

See

- MISRA C++ 2008, 5-0-4
- CERT, INT31-C. - Ensure that integer conversions do not result in lost or misinterpreted data

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