

-  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 578

 Vulnerability 13

 Bug 111

 Security Hotspot 18

 Code Smell 436

 Quick Fix 68

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

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

 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

 Bug
"std::move" and "std::forward" should not be confused  Bug
A call to "wait()" on a "std::condition_variable" should have a condition  Bug
A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast  Bug
Functions with "noreturn" attribute should not return  Bug
RAII objects should not be temporary  Bug
"memcmp" should only be called with pointers to trivially copyable types with no padding  Bug
"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types  Bug
"std::auto_ptr" should not be used  Bug
Destructors should be "noexcept"  Bug

Pointer and reference parameters should be "const" if the corresponding object is not modified

Analyze your code

Code Smell Minor Quick Fix bad-practice misra-c++2008 misra-c2004 misra-c2012

This rule leads to greater precision in the definition of the function interface. The `const` qualification shall be applied to the object pointed to, not to the pointer, since it is the object itself that is being protected.

Noncompliant Code Example

```
void myfunc (    int * param1,  // object is modified
               const int * param2,
               int * param3, // Noncompliant
               int * param4) // Noncompliant
{
    *param1 = *param2 + *param3 + *param4;
}

int main (int argc,
          const char * * argv) // Noncompliant
{
    return argc;
}
```

Compliant Solution

```
void myfunc (    int * param1,  // object is modified
               const int * param2,
               const int * param3,
               const int * param4)
{
    *param1 = *param2 + *param3 + *param4;
}

int main (int argc,
          const char * const * argv)
{
    return argc;
}
```

See

- MISRA C:2004, 16.7 - A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object.
- MISRA C++:2008, 7-1-2 - A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.
- MISRA C:2012, 8.13 - A pointer should point to a const-qualified type whenever possible

Available In: sonarlint | sonarcloud | sonarqube Developer Edition