

- 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

Object declarations should contain no more than 2 levels of pointer indirection

Analyze your code

Code Smell Critical based-on-misra brain-overload pitfall

While they are extraordinarily useful, pointers are not the most intuitive concept in the world. Pointers to pointers are even harder to understand and use correctly. And with each additional level of indirection, pointer variables become more difficult to use correctly. Therefore pointer declarators should be limited to no more than two levels of nesting.

Noncompliant Code Example

```
typedef int * INTPTR;
struct s {
    int ** s1;
    int *** s2; // Noncompliant
};

struct s ** ps1;
struct s *** ps2; // Noncompliant

int ** ( *pfunc1)();
int ** ( **pfunc2)();
int ** (**pfunc3)(); // Noncompliant
int *** ( **pfunc4)(); // Noncompliant

void function( int ** par1,
               int *** par2, // Noncompliant
               INTPTR * par3,
               int * par4[],
               int ** par5[]) // Noncompliant
{
    int ** ptr1;
    int *** ptr2; // Noncompliant
    INTPTR * ptr3;
    int * ptr4[ 10 ];
    int ** ptr5[ 10 ]; //Noncompliant
}
```

Compliant Solution

```
typedef int * INTPTR;
struct s {
    int ** s1;
    int ** s2;
};

struct s ** ps1;
struct s ** ps2;

int ** (*pfunc1)();
int ** (**pfunc2)();
int ** (**pfunc3)();
int ** (**pfunc4)();

void function( int ** par1,
               int ** par2,
               INTPTR * par3,
               int * par4[],
               int * par5[])
{
```

 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

```
int ** ptr1;
int ** ptr2;
INTPTR * ptr3;
int * ptr4[ 10 ];
int * ptr5[ 10 ];
}
```

See

- MISRA C:2004, 17.5 - The declaration of objects should contain no more than 2 levels of pointer indirection
- MISRA C++:2008, 5-0-19 - The declaration of objects shall contain no more than two levels of pointer indirection
- MISRA C:2012, 18.5 - Declarations should contain no more than two levels of pointer nesting

Available In:

sonarlint

sonarcloud

sonarqube

Developer Edition