


































-  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

Size of bit fields should not exceed the size of their types

Analyze your code

 Code Smell  Major  bad-practice

Bit fields allow the developers to declare a class member with a specific size.

However, the size of a bit field is also constrained by its type: even if the specified size is greater than the size of the type, the value of the bit field will not exceed the maximum value of this type. The extra bits will just create unused padding.

The incompatibility of the size of the type with the specified size can have two causes: either the specified size is a typo error (that is the most probable cause) or the developer did not realize the size of the type he chose was too small.

### Noncompliant Code Example

```
class A {
    unsigned int b : 55; // Noncompliant, specified size is gre
};
```

### Compliant Solution

```
class A {
    unsigned int b : 32;
};
```

Or

```
class A {
    unsigned long long int b : 55;
};
```

Available In:

sonarlint  | sonarcloud  | sonarqube  Developer Edition

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