






-  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

Switch statement conditions should not have essentially boolean type

Analyze your code

-  Code Smell
-  Minor 
-  misra-c++2008 misra-c2004 misra-c2012

When there is only a single condition to test, you have the option of using either a switch statement or an if-else if-else statement. For a larger set of potential values, a switch can be easier to read, but when the condition being tested is essentially boolean, then an if/else statement should be used instead.

Noncompliant Code Example

```
_Bool b = p > 0;
switch (b) { // Noncompliant
...
}
switch (x == 0) { // Noncompliant
...
}
```

Compliant Solution

```
_Bool b = p > 0;
if (b) {
...
} else {
...
}
if (x == 0) {
...
} else {
...
}
```

See

- MISRA C:2004, 15.4 - A switch expression shall not represent a value that is effectively Boolean
- MISRA C++ 2008, 6-4-7 - The condition of a switch statement shall not have bool type
- MISRA C:2012, 16.7 - A switch-expression shall not have essentially Boolean type

Available In:

sonarlint  | sonarcloud  | sonarqube  Developer Edition

 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