






-  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 rules578
-  Vulnerability13
-  Bug111
-  Security Hotspot18
-  Code Smell436
-  Quick Fix68


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

Exception specifications should not be used

Analyze your code

 Code Smell  Critical  cppcoreguidelines error-handling clumsy

Exception specifications never really provided any advantages in C++. They have been deprecated since C++11, and removed since C++17 (specification with an exception) and C++20 (empty throw specification). If your code still contains some, you should replace empty `throw()` specifications, with `noexcept` and remove any other specifications.

Noncompliant Code Example

```
void f() throw(); // Noncompliant
void g() throw(std::exception); // Noncompliant
```

Compliant Solution

```
void f() noexcept;
void g();
```

Exceptions

If a derived class overrides a function with a dynamic exception specification, then the derived function is forced to add a compatible exception specification. Such exception specifications are ignored.

See

- [C++ Core Guidelines E.30](#) - Don't use exception specifications

Available In:
 |  |  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