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

C-style array should not be used

Analyze your code

Code Smell | Major | cppcoreguidelines bad-practice clumsy

C-style arrays (such as `int i[10]`) are not very convenient to use:

- They are fixed size (even C VLA are not truly variable size, and they are not supported in C++)
- If the number of elements in the array can vary, it will lead to manual memory allocation (or people will use fixed-size arrays that "should be large enough", which is both a waste of memory and a limitation of the program)
- It is very easy to lose the size of the array since an array passed to a function decays into a pointer

The C++ standard library proposes two types that are better than C-style arrays and together cover all the use cases of C-style arrays:

- For fixed-size arrays, where the memory is on the stack, use `std::array`. It is like a C-style array, except that it has a normal argument passing semantic, and the size is always a part of the type. If `std::array` is not available to you (before C++11), you can roll your own version.
- For variable-size arrays, use `std::vector`. It can be resized and handles memory allocation transparently.
- For character strings, you should use `std::string` instead of arrays of characters.
- For arrays of characters that are not strings (e.g., alphabet, exit codes, keyboard control list) perfer `std::array` or `std::vector` as per the first two bullets.

The rule {rule:cpp:S945} is related to this rule but focuses on passing arguments of an array type. {rule:cpp:S5025} will flag the use of dynamic memory allocation that could be replaced by `std::vector`.

Noncompliant Code Example

```
void f() {
  int a[10]; // Noncompliant
}
```

Compliant Solution

```
void f() {
  std::array<int, 10> a1; // If the size really is a constant
  // Or
  std::vector<int>a2; // For variable size

  auto s = "Hello!"; // Compliant by exception
}
```

Exceptions

This rule will not report the use of C-style arrays in `extern "C"` code (since those arrays are often required here for compatibility with external code) and in the arguments of `main`.

See

- C++ Core Guidelines SL.con.1 - Prefer using STL array or vector instead of a C array

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