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

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

"void *" should not be used in typedefs, member variables, function parameters or return type

Analyze your code

Code Smell Critical cppcoreguidelines based-on-misra

`void*` is a pointer to memory of unknown type, and therefore works outside of the safety net provided by the type system. While it can be useful in a function body to interface with external code, there is no good reason to step out of the robust C++ type system when defining a function, either for the function parameters, or for the function return type. For the same reasons, having a member variable of type `void*` is not recommended.

If you want to work with raw memory buffer, use `unsigned char *` (or `byte *` if your compiler supports it).

If you want to work with different types of data, define a function template and use typed pointers, instead of `void *`. If you want a single object to be able to stores objects of different types, `std::any` can also be a type-safe alternative to `void*`.

If you want to provide to users of an API an opaque type, declare a type and don't provide its definition (like with `FILE*`).

Note that `void*` is commonly used to communicate data of unknown type with C code. This rule will nevertheless raise an issue in this case, but it can be ignored.

Noncompliant Code Example

```
void saveBuffer(void *buffer, size_t size); // Noncompliant
void duplicate(void* destination, size_t count, void *source,
class Process {
  // ...
  void *userData;
};
using UserData = void*; // Noncompliant
```

Compliant Solution

```
void saveBuffer(unsigned char *buffer, size_t size);
template<class T>
void duplicate(T* destination, size_t count, T *source);
class Process {
  // ...
  std::any userData;
};
```

Exceptions

`void*` can be useful when interfacing with C. As such, the rule will ignore `extern "C"` functions, as well as types with standard layout.

See

- C++ Core Guidelines I.4 - Make interfaces precisely and strongly typed
- C++ Core Guidelines T.3 - Use templates to express containers and ranges

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition