sonar RULES Products

- 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

"std::byte" should be used when you need byte-oriented memory access

Analyze your code

Code Smell | Major | since-c++17 clumsy pitfall

C++17 introduced std::byte. It allows you to have byte-oriented access to a memory in a type-safe unambiguous manner. Before, you had to use either `char`, `signed char`, or `unsigned char` to access memory as bytes. The previous approach is error-prone as `char` type allows you to accidentally perform arithmetic operations. Also, it is confusing since `char`, `signed char`, and `unsigned char` are also used to represent actual characters and arithmetic values.

`std::byte` is simply a scoped enumeration with bit-wise operators and a helper function `to_integer<T>` to convert byte object to integral type T.

This rule will detect byte-like usage of `char`, `signed char`, and `unsigned char` and suggest replacing them by `std::byte`.

Noncompliant Code Example

```
void handleFirstByte(char* byte);

void f(int* i) {
  char* c = reinterpret_cast<char*>(i); // Noncompliant
  handleFirstByte(c);
}

unsigned char negate(unsigned char byte) {
  return ~byte; // Noncompliant
}
```

Compliant Solution

```
void handleFirstByte(std::byte* byte);

void f(int* i) {
  std::byte* byte = reinterpret_cast<std::byte*>(i); // Compl
  handleFirstByte(byte);
}

std::byte negate(std::byte byte) {
  return ~byte; // Compliant
}
```

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