



ABAP

APEX Apex

c C

C++

CloudFormation

COBOL COBOL

C# C#

CSS

◯ ⋉ Flex

€60 Go

5 HTML

🎒 Java

Js JavaScript

Kotlin

Kubernetes

Objective C

PP PHP

PL/I

PL/SQL

Python

RPG RPG

Ruby

Scala

Swift

Terraform

Text

тs TypeScript

T-SQL

VB VB.NET

VB6 VB6

XML XML



C++ static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C++ code

All 578 rules Vulnerability 13

R Bug (111)

Security Hotspot

Quick 68 Fix

Analyze your code

Tags

"insert" with "std::set" and

"std::unordered_set"

"emplace" should be prefered over

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

<table-of-contents> Bug

Objects should not be sliced

📆 Bug

Immediately dangling references should not be created

🕀 Bug

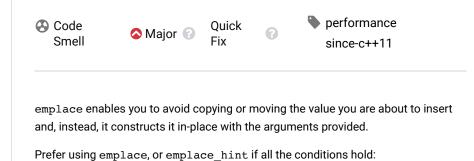
"pthread_mutex_t" should be unlocked in the reverse order they were locked

📆 Bug

"pthread_mutex_t" should be properly initialized and destroyed

📆 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice



erer using emptace, or emptace_ntire if all the conditi

- You are inserting a single value.
- You are constructing a fresh temporary value just to insert it into the set.
- You expect that the key is not in the set.

You should keep the insert in any of the cases below:

- You are inserting multiple values in one shot.
- You are inserting a pre-existing value that is constructed for another purpose.
- You are inserting an object that is cheap to move or to copy (e.g., an integer).
- The key you are inserting is likely to be in the set (in this case by using insert you avoid creating a useless temporary node).

This rule detects calls to insert that lead to the creation of a large temporary object that can be avoided by using the emplace member function.

Noncompliant Code Example

```
struct A {
  int x;
  std::array<std::string, 100> more;// Expensive to copy or m
public:
  A(int x, const std::string& more) : x(x), more({more}) {}
  bool operator<(A const &other) const {
    return x < other.x;
  }
};
std::array<std::string, 3> strs = {"big brown fox", "little k
void f() {
  std::set<A> set;
  for (int i = 0; i < 1'000'000; ++i) {
    set.insert(A{i, strs[i%3]});// Noncompliant
  }
}</pre>
```

Compliant Solution



"std::move" and "std::forward" should not be confused



A call to "wait()" on a "std::condition_variable" should have a condition



A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast



Functions with "noreturn" attribute should not return



RAII objects should not be temporary



"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
```

```
struct A {
  int x;
  std::array<std::string, 100> more;// Expensive to copy or m
public:
  A(int x, const std::string& more) : x(x), more({more}) {}
  bool operator<(A const &other) const {
    return x < other.x;
  }
};
std::array<std::string, 3> strs = {"big brown fox", "little k
void f() {
  std::set<A> set;
```

© 2008-2022 SonarSource's A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE! SONARLINATES NARQUBE and SONARCEOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved.

Privacy Policy

Exceptions

You should keep insert for exception safety if your key type is a smart pointer and the argument is a new expression.

Available In:

sonarlint o | sonarcloud o | sonarqube | Developer Edition