



ABAP Apex

С

C++

CloudFormation

COBOL

C#

CSS

Flex

Go =GO

5 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

578 ΑII 6 Vulnerability 13 rules

R Bug (111)

o Security Hotspot

⊗ Code (436)

Quick 68 Fix

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

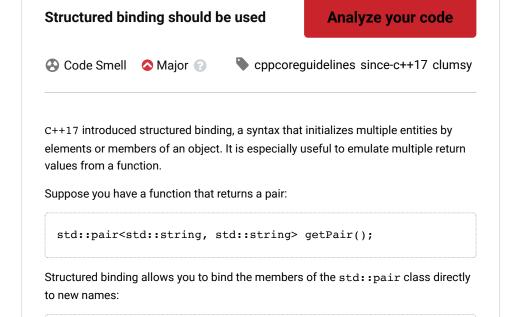
"pthread_mutex_t" should be unlocked in the reverse order they were locked

📆 Bug

"pthread_mutex_t" should be properly initialized and destroyed

📆 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice



The names firstName and lastName are called structured bindings. As you can see, structured binding makes the code more readable as they allow binding values to names that carry information about their purpose.

Structured binding works with:

Raw arrays, by binding a name to each element

auto [firstName, lastName] = getPair();

- · Any type that has a tuple-like API
- Classes and structures where all non-static data member are publicly accessible

This rule will detect places where std::pair and std::tuple can be effortlessly replaced by a structured binding.

Noncompliant Code Example

```
void printingMap(const std::map<int, std::string>& map) {
  for (const auto& elem : map) { // Noncompliant
    std::cout << elem.first << ": " << elem.second << "\n";</pre>
}
```

Compliant Solution

```
void printingMap(const std::map<int, std::string>& map) {
  for (const auto& [key, value] : map) { // Compliant
    std::cout << key << ": " << value << "\n";
}
```

See

• C++ Core Guidelines F.21 - To return multiple "out" values, prefer returning a struct or tuple

Available In:

sonarlint 😊 | sonarcloud 🙆 | sonarqube | Developer Edition

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective

 $\begin{array}{c} \text{owners. All rights are expressly reserved.} \\ \underline{\text{Privacy Policy}} \end{array}$

∄ 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
fit 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
fit Bug
"std::auto_ptr" should not be used
fit Bug
Destructors should be "noexcept"
👬 Bug