



ABAP

Apex Apex

C C

© C++

CloudFormation

COBOL COBOL

C# C#

∃ CSS

⊠ Flex

GO Go

5 HTML

Java

Js JavaScript

Kotlin

Kubernetes

Objective C

PHP

PL/I

PL/SQL

🦆 Python

RPG RPG

Ruby

Scala

Swift

Terraform

■ Text

Ts TypeScript

T-SQL

VB VB.NET

VB6 VB6

xmL XML

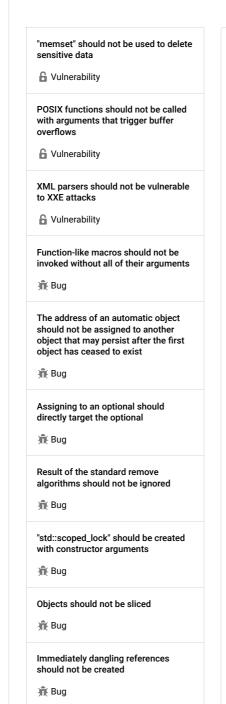


C++ static code analysis

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

All 578 rules Support Security 18 Security

Tags



"pthread_mutex_t" should be unlocked

in the reverse order they were locked

"pthread_mutex_t" should be properly

Bug

Use "std::variant" instead of unions with non-trivial types.

Analyze your code

Search by name.

In order to save memory, unions allow you to use the same memory to store objects from a list of possible types as long as one object is stored at a time.

In C and in C++ prior to C++11, unions are restricted to trivial types.

Starting from C++11, it is possible to use unions with non-trivial types with the following limitations:

- You have to manually handle the lifetime of the active member, using placement new and explicit object destruction.
- You have to define special members like destructor and copy-constructor while taking into consideration the active member.

In some cases, code that fails to perfectly follow those rules may still compile, but lead to memory corruption.

C++17 introduced std::variant which can replace unions while removing this burden and the associated risk. As a safer and more readable alternative, they should be preferred.

Noncompliant Code Example

```
#include <new> // Required for placement 'new'.
#include <string>
#include <iostream>
using namespace std;
struct IntOrString {
  enum {holdsInt, holdsString} currentAlternative;
  union {
   int z:
   string s; // Noncompliant: non-trivial type in Union
  };
  IntOrString() : currentAlternative{holdsInt} {
  }
  IntOrString(char const *s) : currentAlternative{holdsString
   new(&s) string(s);
  }
  IntOrString(IntOrString const &src) : currentAlternative(sr
     if (currentAlternative == holdsString) {
         new(&s) string(src.s);
  IntOrString &operator=(IntOrString &&) = delete;
  ~IntOrString() {
     if (currentAlternative == holdsString) {
          s.~string():
 }
};
void stringize(IntOrString &ios) {
```

initialized and destroyed

👬 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked

👬 Bug

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

🕕 Bug

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

```
if (ios.currentAlternative == IntOrString::holdsString) {
   new (&ios.s) string(std::to_string(ios.z));
int main() {
 IntOrString ios;
 auto copy = ios;
 ios.z = 12;
 stringize(ios);
 std::cout<< ios.s << "\n";
```

Compliant Solution

```
#include <variant>
#include <iostream>
#include <string>
using namespace std;
using IntOrString = variant<int, string>;
void stringize(IntOrString &ios) {
   if(auto i = get_if<int>(&ios)) {
       ios = to_string(*i);
}
int main() {
   IntOrString ios = 12;
   auto copy = ios;
   stringize(ios);
    cout << std::get<string>(ios) << '\n';</pre>
}
```

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 owners. All rights are expressly reserved. Privacy Policy