



SAP ABAP

Apex Apex

C C

C++

CloudFormation

COBOL COBOL

C# C#

E CSS

⊠ Flex

GO Go

HTML

🆺 Java

_{JS} JavaScript

Kotlin

Kubernetes

SObjective C

PP 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 Vulnerability 13

R Bug (111)

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

<table-of-contents> Bug

Objects should not be sliced

👬 Bug

Immediately dangling references should not be created

🕀 Bug

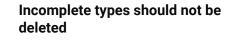
"pthread_mutex_t" should be unlocked in the reverse order they were locked

📆 Bug

"pthread_mutex_t" should be properly initialized and destroyed

📆 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice



Analyze your code

When calling delete on an object of incomplete type, the calling code does not have enough information to do the action properly (it does not know if this object has a trivial or a nontrivial destructor, if it has overloaded the delete operator...). Therefore, deleting a pointer to such an object can lead to undefined behavior.

Noncompliant Code Example

```
class Body;

class Handle {
  public:
    Handle();
    ~Handle() {
      delete impl; // Noncompliant, Body is incomplete
    }
  private:
    Body * impl;
};
```

Compliant Solution

```
// In header file
class Body;

class Handle {
  public:
    Handle();
    ~Handle();
    // Add other special member functions to respect the rule
  private:
    Body * impl;
};

// In implementation file
#include "Handle.h"
#include "Body.h" // Now Body is complete

Handle::~Handle(){
  delete impl; // Compliant, at this point "Body" is a comple
}
```

Or, with modern resource handling:

```
// In header file
class Body;

class Handle {
  public:
    Handle();
    ~Handle();
  private:
    std::unique_ptr<Body> impl; //Compliant
};

// In implementation file
#include "Handle.h"
```

📆 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 Rug 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 Rug Bug "std::auto_ptr" should not be used

📆 Bug

📆 Bug

Destructors should be "noexcept"

```
#include "Body.h" // Now Body is complete

Handle::Handle() : impl{new Body{}} {}

Handle::~Handle() = default; // since "Body" is complete, it

Available In:

sonarlint o Sonarcloud o 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