



ABAP

Apex

С

C++

CloudFormation

COBOL

C#

CSS

Flex

Go =GO

HTML 5

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

ΑII 578 rules

6 Vulnerability 13

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 Functions should not contain too many return statements

Analyze your code

brain-overload

Having too many return statements in a function increases the function's essential complexity because the flow of execution is broken each time a return statement is encountered. This makes it harder to read and understand the logic of the function.

The way of counting the return statements is aligned with the way we compute **Cognitive Complexity.**

"Under Cyclomatic Complexity, a switch is treated as an analog to an if-else if chain [...] but from a maintainer's point of view, a switch - which compares a single variable to an explicitly named set of literal values - is much easier to understand than an if-else if chain because the latter may make any number of comparisons, using any number of variables and values. "

As a consequence, all the return statements located at the top level of case statements (including default) of a same switch statement count all together as 1.

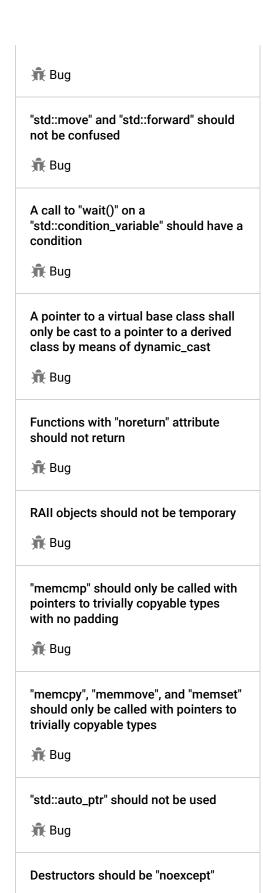
```
// this counts as 1 return
int fun() {
  switch(variable) {
  case value1:
    return 1;
  case value2:
    return 2;
  default:
    return 3;
}
```

Noncompliant Code Example

With the default threshold of 3:

```
// this counts as 3 returns
int fun() {
 if (condition1) {
    return 1;
 } else {
    if (condition2) {
      return 0;
   } else {
      return 1;
 return 0;
```

```
// this counts as 3 returns
int fun() {
  switch(variable) {
  case value1:
    if(condition1) {
      return 1;
    } else {
      return -1;
  default:
    return 2;
```



📆 Bug

Available In:

sonarlint image | sonarcloud to | 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