



ABAP

APEX Apex

C C

© C++

CloudFormation

COBOL COBOL

C# C#

CSS

X Flex

GO Go

THIML

🔮 Java

Js JavaScript

Kotlin

Kubernetes

Ó 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 311 vulnerability 13

♣ Bug 74

Security
Hotspot

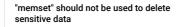
Enums should be consistent with

the bit fields they initialize

⇔ Code Smell 206

O Quick 14 Fix

Tags Search by name.



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

<table-of-contents> 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

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

in∈ Bua

"pthread_mutex_t" should be properly initialized and destroyed

🕕 Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

👬 Bug

Functions with "noreturn" attribute should not return

👬 Bug

"memcmp" should only be called with pointers to trivially copyable types with no padding

📆 Bug

18

Analyze your code

Bit fields can only have integral or enumeration type. If it is quite straightforward to check if an integral type can initialize a bit field, it is however trickier with an enum type: the bit field has to be wide enough to store all the possible values of the enum.

In addition to this, the signedness of the enum should be consistent with the signedness of the bit field:

- an unsigned bit field can not be initialized with a signed enum type
- a signed bit field uses one bit to store the sign and this needs to be taken into account while comparing the size of the enum type with the size of the bit field.

Noncompliant Code Example

```
enum Color {
  BLUE = 16
} myColor;
enum Fruit {
 ORANGE = 1,
  APPLE = 2
} myFruit;
struct BitStructForColor {
   unsigned int b : 2;
struct BitStructForFruit {
   signed int b : 2;
};
void f(BitStructForColor &bColorStruct, BitStructForFruit &
 bColorStruct.b = myColor; // Noncompliant, myColor is too w
  bFruitStruct.b = myFruit; // Noncompliant, one bit of the b
}
```

Compliant Solution

```
enum Color {
   BLUE = 16
} myColor;
enum Fruit {
   ORANGE = 1,
   APPLE = 2
} myFruit;
struct BitStructForColor {
    unsigned int b : 5;
};
struct BitStructForFruit {
```

Stack allocated memory and nonowned memory should not be freed

🕕 Bug

Closed resources should not be accessed

📆 Bug

Dynamically allocated memory should be released

🛊 Bug

Freed memory should not be used

```
signed int b : 3;
};

void f(BitStructForColor &bColorStruct, BitStructForFruit & bColorStruct.b = myColor; bFruitStruct.b = myFruit;
}

Available In:

sonarlint ⋈ sonarcloud ் sonarqube 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