



ABAP

APEX Apex

C C

© C++

CloudFormation

COBOL COBOL

C#

**∃** CSS

X Flex

**-co** Go

**⋾** 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 311 rules & Vulnerability 13

**∰** Bug 74

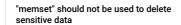
74

Security 18 Hotspot ⊗ Code 206 Smell

O Quick 14

Tags V Search by name... Q

symbolic-execution multi-threading



■ 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

"pthread\_mutex\_t" should be unlocked in the reverse order they were locked

₩ Buc

"pthread\_mutex\_t" should be properly initialized and destroyed

📆 Bug

"pthread\_mutex\_t" should not be consecutively locked or unlocked twice

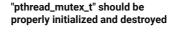
👬 Bug

Functions with "noreturn" attribute should not return

👬 Bug

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

🖷 Bug



# Bug Blocker

Analyze your code

 ${\it Mutexes} \ {\it are} \ {\it synchronization} \ {\it primitives} \ {\it that} \ {\it allow} \ {\it to} \ {\it manage} \ {\it concurrency}.$ 

Their use requires following a well-defined life-cycle.

- Mutexes need to be initialized (pthread\_mutex\_init) before being used. Once
  it is initialized, a mutex is in an unlocked state.
- Mutexes need to be destroyed (pthread\_mutex\_destroy) to free the associated internal resources. Only unlocked mutexes can be safely destroyed.

Before initialization or after destruction, a mutex is in an uninitialized state.

About this life-cycle, the following patterns should be avoided as they result in an undefined behavior:

- trying to initialize an initialized mutex
- trying to destroy an initialized mutex that is in a locked state
- trying to destroy an uninitialized mutex
- trying to lock an uninitialized mutex
- trying to unlock an uninitialized mutex

In C++, it is recommended to wrap mutex creation/destruction in a RAII class, as well as mutex lock/unlock. Those RAII classes will perform the right operations, even in presence of exceptions.

## Noncompliant Code Example

```
pthread mutex t mtx1:
void bad1(void)
  pthread_mutex_init(&mtx1);
 pthread_mutex_init(&mtx1);
void bad2(void)
  pthread_mutex_init(&mtx1);
 pthread_mutex_lock(&mtx1);
 pthread_mutex_destroy(&mtx1);
void bad3(void)
 pthread_mutex_init(&mtx1);
  pthread_mutex_destroy(&mtx1);
  pthread_mutex_destroy(&mtx1);
void bad4(void)
  pthread_mutex_init(&mtx1);
  pthread_mutex_destroy(&mtx1);
 pthread_mutex_lock(&mtx1);
```

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

```
void bad5(void)
 pthread_mutex_init(&mtx1);
 pthread_mutex_destroy(&mtx1);
 pthread_mutex_unlock(&mtx1);
```

## **Compliant Solution**

```
pthread_mutex_t mtx1;
void okl(void)
  pthread_mutex_init(&mtx1);
 pthread_mutex_destroy(&mtx1);
void ok2(void)
 pthread_mutex_init(&mtx1);
 pthread_mutex_lock(&mtx1);
 pthread_mutex_unlock(&mtx1);
  pthread_mutex_destroy(&mtx1);
```

• The Open Group pthread\_mutex\_init, pthread\_mutex\_destroy

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