

Go =GO

5 HTML

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



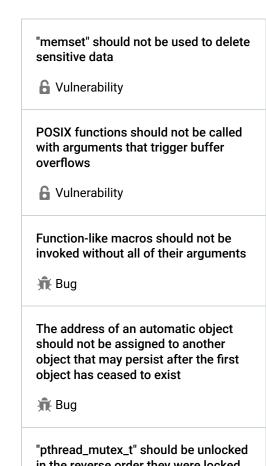
Objective C static code analysis

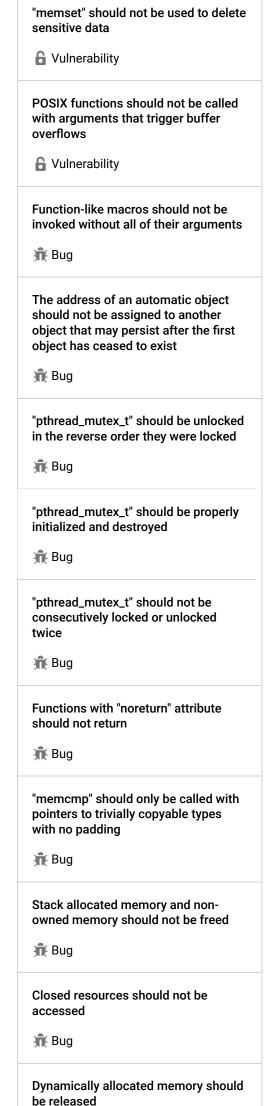
Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your OBJECTIVE C code

o Security ⊗ Code (212) O Quick 13 Fix ΑII 315 **R** Bug (75) 6 Vulnerability 10 rules Hotspot

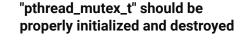
Tags

🙀 Bug 🕕 Blocker 🕝





📆 Bug



Analyze your code

Search by name...

symbolic-execution multi-threading

Mutexes are synchronization primitives that allow to manage 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

Freed memory should not be used

📆 Bug

Memory locations should not be released more than once

📆 Bug

Memory access should be explicitly bounded to prevent buffer overflows

📆 Bug

Printf-style format strings should not lead to unexpected behavior at runtime

📆 Bug

Recursion should not be infinite

👬 Bug

Resources should be closed

👬 Bug

Hard-coded credentials are securitysensitive

Security Hotspot

"goto" should jump to labels declared later in the same function

Code Smell

Only standard forms of the "defined" directive should be used

Code Smell

Switch labels should not be nested inside non-switch blocks

Code Smell

```
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);
void bad5(void)
```

© 2000:2022:35 Dat Country of their respective pthread_mutex_units are the property of their respective pthread_mutex_units. All other trademarks and copyrights are the property of their respective pthread_mutex_units. All office are expressly reserved.

Privacy Policy

Compliant Solution

```
pthread_mutex_t mtx1;

void ok1(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_unlock(&mtx1);
   pthread_mutex_destroy(&mtx1);
}
```

See

• The Open Group pthread_mutex_init, pthread_mutex_destroy

Available In:

sonarcloud 🚳 | sonarqube | Developer Edition