sonar RULES

Products

- Secrets
- ABAP
- Apex
- C
- C++
- CloudFormation
- COBOL
- C#
- CSS
- Flex
- Go
- 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

All rules 315 | Vulnerability 10 | Bug 75 | Security Hotspot 18 | Code Smell 212 | Quick Fix 13

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

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
Bug

"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

Stack allocated memory and non-owned memory should not be freed
Bug

Closed resources should not be accessed
Bug

Dynamically allocated memory should be released
Bug

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

Analyze your code

Bug Blocker unpredictable

The function `memcmp` can only be used for objects of trivially copyable types. This includes scalar types, arrays, and trivially copyable classes.

A class type is trivially copyable if:

- One or more of the following methods is trivial and the rest are deleted: copy constructor, move constructor, copy assignment operator, and move assignment operator,
- It has a trivial, non-deleted destructor.

Additionally, if the type contains padding, some of its bits might be non-representative, and a strict comparison of raw memory contents might lead to the mistaken belief that two identical objects are actually different.

Noncompliant Code Example

```
class Shape { // Trivially copyable, but will contain padding
public:
  bool visible;
  int x;
  int y;
};

bool isSame(Shape *s1, Shape *s2)
{
    return memcmp(s1, s2, sizeof Shape) == 0; // Noncompliant
}
```

Compliant Solution

```
class Shape {
public:
  bool visible;
  int x;
  int y;
};

bool operator==(Shape const &s1, Shape const &s2) {
  return s1.visible == s2.visible && s1.x == s2.x && s1.y ==
}

bool isSame(Shape *s1, Shape *s2)
{
    return (*s1) == (*s2);
}
```

Available In:
sonarcloud sonarqube Developer Edition

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 security-sensitive

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