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

Reserved identifiers and functions in the C standard library should not be defined or declared

Analyze your code

Code Smell | Blocker | based-on-misra bad-practice cert

Defining or declaring identifiers with reserved names may lead to undefined behavior. Similarly, defining macros, variables or functions/methods with the same names as functions from the C standard library is likely to lead to unexpected results.

Additionally, such identifiers have the potential to thoroughly confuse people who are unfamiliar with the code base, possibly leading them to introduce additional errors. Therefore reserved words and the names of C standard library functions should not be used as identifiers.

This rule applies to:

- `defined`
- C standard library function names
- identifiers that contain two consecutive underscores
- identifiers that begin with an underscore, followed by an uppercase letter
- identifiers in the global namespace that start with an underscore

Noncompliant Code Example

```
#ifndef _MY_FILE
#define _MY_FILE   // Noncompliant: starts with '_'

#define FIELD__VAL(field) ##field // Noncompliant: contains "

int free(void *pArg, int len) {  // Noncompliant: free is a s
  int __i; // Noncompliant: starts with "__"
  //...
}
#endif
```

Compliant Solution

```
#ifndef MY_FILE
#define MY_FILE

#define FIELD_VAL(field) ##field

int clean(void *pArg, int len) {
  int i;
  //...
}
#endif
```

See

- MISRA C:2004, 20.1 - Reserved identifiers, macros and functions in the standard library, shall not be defined redefined or undefined.
- MISRA C++:2008, 17-0-1 - Reserved identifiers, macros and functions in the standard library shall not be defined, redefined, or undefined.
- MISRA C:2012, 21.2 - A reserved identifier or macro name shall not be declared
- CERT, DCL37-C. - Do not declare or define a reserved identifier
- CERT, DCL51-CPP. - Do not declare or define a reserved identifier

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