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

Exceptions should not be used

Analyze your code

Code Smell Major

While exceptions are a common feature of modern languages, there are several reasons to potentially avoid them:

- They make the control flow of a program difficult to understand, because they introduce additional exit points.
- The use of exceptions in new code can make that code difficult to integrate with existing, non-exception-safe code.
- They add to the size of each binary produced, thereby increasing both compile time and final executable size.
- They may incur a small performance penalty.
- The time required to handle an exception is not easy to assess, which makes them difficult to use for hard real-time applications.

This rule raises an issue when:

- an exception is `thrown`
- a `try-catch` block is used
- an exception specification (`throw(xxx)`) is present.

Noncompliant Code Example

This C++ code example also applies to Objective-C.

```
double myfunction(char param) throw (int); // Noncompliant
void f {
  try // Noncompliant
  {
    do_something();
    throw 1; // Noncompliant
  }
  catch (...)
  {
    // handle exception
  }
}
```

Compliant Solution

```
double myfunction(char param) noexcept;
bool f {
  if (!do_something()); {
    // Handle the situation
    return false;
  }
  // Rest of the code
  return true;
}
```

Exceptions

`noexcept` specifications are ignored, because even if you choose not to use exceptions in your code, it's important to decorate as `noexcept` certain functions (for instance, move constructors that do not `throw`). This decoration can be detected by type traits, and some meta-programming techniques rely on this information.

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

Available In:

sonarcloud | sonarqube Developer Edition