






































-  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

"memcpy" 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 ignored

Analyze your code

 Code Smell  Minor  cwe error-handling owasp suspicious

When exceptions occur, it is usually a bad idea to simply ignore them. Instead, it is better to handle them properly, or at least to log them.

### Noncompliant Code Example

```
void save() {
    try {
        saveDocument();
    } catch (const std::exception& ex) {
    }
}
```

### Compliant Solution

```
void save() {
    try {
        saveDocument();
    } catch (const std::exception& ex) {
        log << "Exception while saving the document: " << ex.what();
    }
}
```

### See

- [OWASP Top 10 2021 Category A9](#) - Security Logging and Monitoring Failures
- [OWASP Top 10 2017 Category A10](#) - Insufficient Logging & Monitoring
- [MITRE, CWE-390](#) - Detection of Error Condition Without Action

Available In:

**sonarcloud**  | **sonarqube**  Developer Edition

<div>Freed memory should not be used</div> <div> Bug</div>
<div>Memory locations should not be released more than once</div> <div> Bug</div>
<div>Memory access should be explicitly bounded to prevent buffer overflows</div> <div> Bug</div>
<div>Printf-style format strings should not lead to unexpected behavior at runtime</div> <div> Bug</div>
<div>Recursion should not be infinite</div> <div> Bug</div>
<div>Resources should be closed</div> <div> Bug</div>
<div>Hard-coded credentials are security-sensitive</div> <div> Security Hotspot</div>
<div>"goto" should jump to labels declared later in the same function</div> <div> Code Smell</div>
<div>Only standard forms of the "defined" directive should be used</div> <div> Code Smell</div>
<div>Switch labels should not be nested inside non-switch blocks</div> <div> Code Smell</div>