

-  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

Macros should not be used as replacement to "typedef" and "using"

Analyze your code

 Code Smell  Minor  bad-practice cert pitfall

C provides a way of defining or aliasing a type through `typedef`. On top of it, C++ adds `using` that can do the same and more.

Using a macro to define a type is inferior to the previous ways for two reasons:

- macros cannot be enclosed into scopes. Or at least, doing so is cumbersome and error-prone as in that case, the macro needs to be defined and undefined manually.
- macros are handled by the preprocessor and are not understood from the compiler. They can easily pollute the code in places where types are not expected. `typedef` and `using` are known to the compiler to define types and can be more strictly checked.

As a result, macros should not be used as a replacement to `typedef` or `using`.

Noncompliant Code Example

```
#define UINT unsigned int // Noncompliant
#define INT int // Noncompliant
UINT uabs( INT i );
```

Compliant Solution

```
typedef unsigned int UINT;
typedef int INT;
UINT uabs( INT i );
```

or

```
using UINT = unsigned int;
using INT = int;
UINT uabs( INT i );
```

See

- [CERT, PRE03-C](#). - Prefer typedefs to defines for encoding non-pointer types

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>