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

# C static code analysis

Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your C code

All rules 311
Vulnerability 13
Bug 74
Security Hotspot 18
Code Smell 206
Quick Fix 14

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

XML parsers should not be vulnerable to XXE attacks

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

**Freed memory should not be used**

# "goto" should jump to labels declared later in the same function

Analyze your code

Code Smell | Blocker | based-on-misra | pitfall

Unconstrained use of `goto` can lead to programs that are extremely difficult to comprehend and analyse. For C++, it can also lead to the program exhibiting unspecified behavior.

However, in many cases a total ban on `goto` requires the introduction of flags to ensure correct control flow, and it is possible that these flags may themselves be less transparent than the `goto` they replace.

Therefore, the restricted use of `goto` is allowed where that use will not lead to semantics contrary to developer expectations. "Back" jumps are prohibited, since they can be used to create iterations without using the well-defined iteration statements supplied by the core language.

### Noncompliant Code Example

```
int f() {
  int j = 0;
L1:
  ++j;
  if (10 == j) {
    goto L2;          // forward jump ignored
  }
  // ...
  goto L1;            // Noncompliant
L2:
  return ++j;
}
```

### Compliant Solution

```
int f() {
  for (int j = 0; j < 11; j++) {
    // ...
  }
  return ++j;
}
```

### See

- MISRA C++:2008, 6-6-2 - The goto statement shall jump to a label declared later in the same function body
- MISRA C:2012, 15.2 - The goto statement shall jump to a label declared later in the same function

Available In:

sonarlint | sonarcloud | sonarqube Developer Edition