sonar RULES

Products

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

Two branches in a conditional structure should have exactly the same implementation

Analyze your code

Code Smell Major design suspicious

Having two `cases` in a `switch` statement or two branches in an `if` chain with the same implementation is at best duplicate code, and at worst a coding error. If the same logic is truly needed for both instances, then in an `if` chain they should be combined, or for a `switch`, one should fall through to the other.

Noncompliant Code Example

```
switch (i) {
  case 1:
    doFirstThing();
    doSomething();
    break;
  case 2:
    doSomethingDifferent();
    break;
  case 3:  // Noncompliant; duplicates case 1's implementatio
    doFirstThing();
    doSomething();
    break;
  default:
    doTheRest();
}

if (a >= 0 && a < 10) {
  doFirstThing();
  doTheThing();
}
else if (a >= 10 && a < 20) {
  doTheOtherThing();
}
else if (a >= 20 && a < 50) {
  doFirstThing();
  doTheThing();  // Noncompliant; duplicates first condition
}
else {
  doTheRest();
}
```

Exceptions

Blocks in an `if` chain that contain a single line of code are ignored, as are blocks in a `switch` statement that contain a single line of code with or without a following `break`.

```
if (a == 1) {
  doSomething();  //no issue, usually this is done on purpose
} else if (a == 2) {
  doSomethingElse();
} else {
  doSomething();
```

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

```
}
```

But this exception does not apply to `if` chains without `else`-s, or to `switch`-es without default clauses when all branches have the same single line of code. In case of `if` chains with `else`-s, or of `switch`-es with default clauses, rule {rule:cpp:S3923} raises a bug.

```
if (a == 1) {
  doSomething();  //Noncompliant, this might have been done o
} else if (a == 2) {
  doSomething();
}
```

Available In:

sonarlint sonarcloud sonarqube Developer Edition