

- 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 **578**

Vulnerability **13**

Bug **111**

Security Hotspot **18**

Code Smell **436**

Quick Fix **68**

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

Assigning to an optional should directly target the optional

Bug

Result of the standard remove algorithms should not be ignored

Bug

"std::scoped\_lock" should be created with constructor arguments

Bug

Objects should not be sliced

Bug

Immediately dangling references should not be created

Bug

"pthread\_mutex\_t" should be unlocked in the reverse order they were locked

Bug

"pthread\_mutex\_t" should be properly

### Dynamically allocated memory should be released

Analyze your code

Bug Blocker

cwe symbolic-execution leak denial-of-service cert

Memory allocated dynamically with `calloc(...)`, `malloc(...)`, `realloc(...)` or `new` should be released when it's not needed anymore. Failure to do so will result in a memory leak that could bring the box to its knees.

This rule raises an issue when memory is allocated and not freed in the same function. Allocated memory is ignored if a pointer to it is returned to the caller or stored in a structure that's external to the function.

#### Noncompliant Code Example

```
int fun() {
    char* name = (char *) malloc (size);
    if (!name) {
        return 1;
    }
    // ...
    return 0; // Noncompliant, memory pointed by "name" has not
}
```

#### Compliant Solution

```
int fun() {
    char* name = (char *) malloc (size);
    if (!name) {
        return 1;
    }
    // ...
    free(name);
    return 0;
}
```

#### See

- [MITRE, CWE-401](#) - Improper Release of Memory Before Removing Last Reference ('Memory Leak')
- [MEM00-C](#) - Allocate and free memory in the same module, at the same level of abstraction
- [CERT, MEM31-C](#) - Free dynamically allocated memory when no longer needed




Available in:

sonarlint

sonarcloud

sonarqube

Developer  
Edition

<div>initialized and destroyed</div> <div> Bug</div>
<div>"pthread_mutex_t" should not be consecutively locked or unlocked twice</div> <div> Bug</div>
<div>"std::move" and "std::forward" should not be confused</div> <div> Bug</div>
<div>A call to "wait()" on a "std::condition_variable" should have a</div>