

-  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 initialized and destroyed

Bug

"pthread_mutex_t" should not be consecutively locked or unlocked twice

Appropriate arguments should be passed to UNIX/POSIX functions

Analyze your code

Code Smell Critical ? symbolic-execution suspicious

UNIX/POSIX functions can have undefined behavior if they are not called correctly. More specifically:

- allocation size of `calloc`, `malloc`, `realloc`, `reallocf`, `alloca` and `valloc` should be strictly positive
- `open` and `openat` should be called with a flag that contains one access mode: `O_RDONLY`, `O_WRONLY`, or `O_RDWR`
- `open` and `openat` with flag `O_CREAT` should be called with a third argument
- flag `O_EXCL` should be used with `O_CREAT`
- first argument of `pthread_once` should not have automatic storage duration and should be initialized by `PTHREAD_ONCE_INIT`

Noncompliant Code Example

```
int res = open(file, O_CREAT); // Noncompliant, flag O_CREAT

void *mem = alloca(0); // Noncompliant, allocation of 0 bytes

extern void initialize();
pthread_once_t pthread = PTHREAD_ONCE_INIT;
pthread_once(&pthread, initialize); // Noncompliant, do not p
```

Available In:

sonarlint

sonarcloud

sonarqube

Developer Edition

© 2008-2022 SonarSource S.A., Switzerland. All content is copyright protected. SONAR, SONARSOURCE, SONARLINT, SONARQUBE and SONARCLOUD are trademarks of SonarSource S.A. All other trademarks and copyrights are the property of their respective owners. All rights are expressly reserved. [Privacy Policy](#)

 Bug
<p>"std::move" and "std::forward" should not be confused</p>  Bug
<p>A call to "wait()" on a "std::condition_variable" should have a condition</p>  Bug
<p>A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast</p>  Bug
<p>Functions with "noreturn" attribute should not return</p>  Bug
<p>RAII objects should not be temporary</p>  Bug
<p>"memcmp" should only be called with pointers to trivially copyable types with no padding</p>  Bug
<p>"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types</p>  Bug
<p>"std::auto_ptr" should not be used</p>  Bug
<p>Destructors should be "noexcept"</p>  Bug