- 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

Use "make_unique" and "make_shared" to construct "unique_ptr" and "shared_ptr"

Analyze your code

Code Smell | Major | Quick Fix | cppcoreguidelines since-c++11

Prefer `make_unique` and `make_shared` over explicitly calling the constructor of `unique_ptr` and `shared_ptr`, they are more concise since they don't require specifying the type multiple times and they eliminate the need to use new!

Exception-Safety

While `make_unique` and `make_shared` are exception-safe, complex construction of `unique_ptr` and `shared_ptr` might not be.

This is because C++ allows arbitrary order of evaluation of subexpressions.

Consider this example:

```
f(unique_ptr<Lhs>(new Lhs()), throwingFunction());
```

This scenario can happen:

1. Memory is allocated for `Lhs`
2. `Lhs` object is constructed
3. `throwingFunction` is called before the `unique_ptr` construction
4. `throwingFunction` throws an exception.
5. The constructed `Lhs` object is leaked since the `unique_ptr` isn't constructed yet

Note: This scenario can only happen before `C++17`. the new standard states that each argument needs to be fully evaluated before the evaluation of the other arguments. In this case, the explicit construction of `unique_ptr` and `shared_ptr` is exception-safe.

Performance

While `make_unige()` doesn't have an impact on performance, `make_shared()` does.

`make_shared()` performs one heap-allocation. While constructing `shared_ptr()` explicitly will require two: one for the object being managed and the other for the control block that stores data about the ref-counts and the `shared_ptr()` deleter.

Noncompliant Code Example

```
std::unique_ptr<MyClass> uniqueP(new MyClass(42)); // Noncomp
std::shared_ptr<MyClass> sharedP(new MyClass(42)); // Noncomp
```

Compliant Solution

```
auto uniqueP = std::make_unique<MyClass>(42);
auto sharedP = std::make_shared<MyClass>(42);
std::unique_ptr<std::FILE, std::function<void(std::FILE*)>> f
  fopen("example.txt", "r"),
  [](FILE* inFile) { fclose(inFile); }); // compliant: custom
```

Exceptions

This rule ignores code that uses features not supported by `make_shared` or `make_unique`:

- `make_shared` and `make_unique`: using custom deleters,

Bug

"std::move" and "std::forward" should not be confused

Bug

A call to "wait()" on a "std::condition_variable" should have a condition

Bug

A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast

Bug

Functions with "noreturn" attribute should not return

Bug

RAII objects should not be temporary

Bug

"memcmp" should only be called with pointers to trivially copyable types with no padding

Bug

"memcpy", "memmove", and "memset" should only be called with pointers to trivially copyable types

Bug

"std::auto_ptr" should not be used

Bug

Destructors should be "noexcept"

Bug

- `make_shared` and `make_unique`: calling placement-new, i.e. version of `new` with arguments, like `new(std::nothrow)`
- `make_shared` only: using operator `new` provided by class
- `make_shared` before C++20: allocating arrays

See

- C++ Core Guidelines C.150 - Use make_unique() to construct objects owned by unique_ptrs
- C++ Core Guidelines C.151 - Use make_shared() to construct objects owned by shared_ptrs

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