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

---

"std::bit_cast" should be used instead of union type-punning

[**Analyze your code**]

🐛 Bug   ⬥ Major   ⑦   🏷 since-c++20 symbolic-execution bad-practice pitfall

C++20 introduced the `std::bit_cast` function. It standardizes the diverse and sub-optimal approaches of reinterpreting a value as a different type of the same length preserving its binary representation.

One of the superseded solutions, know as "union type-punning", is to use a `union` with two members with types corresponding to the source and the target types of the cast. The operation is performed by saving the value in the member with source type and then reading the value of the target type. Despite being allowed in C, this operation has undefined behavior according to C++ standard and should be replaced by either `std::bit_cast` (or `std::memcpy`).

This rule raises an issue on any use of a `union` that should be replaced with `std::bit_cast`.

Noncompliant Code Example

```
float fastInvSqrt(float number)
  constexpr float threehalfs = 1.5F;
  const float x2 = number * 0.5F;

  union {
      float f;
      uint32_t i;
  } conv;
  conv.f = number;
  conv.i = 0x5f759df - (conv.i >> 1);  // Noncompliant: unde
  conv.f *= threehalfs - (x2 * conv.f * conv.f); // Noncompli
  return conv.f;
}
```

Compliant Solution

```
float fastInvSqrt(float number) {
  constexpr float threehalfs = 1.5F;
  const float x2 = number * 0.5F;

  auto i = std::bit_cast<std::uint32_t>(number);
  i = 0x5f759df - (i >> 1);
  auto result = std::bit_cast<float>(i);
  result *= threehalfs - (x2 * result * result);
  return result;
}
```

See

- {rule:cpp:S6181} - replacing `std::memcpy` with `std::bit_cast`.

Available In:

sonarlint ☺ | sonarcloud ☁ | sonarqube ⬡ Developer Edition

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