ERR61-CPP. Catch exceptions by lvalue reference

When an exception is thrown, the value of the object in the throw expression is used to initialize an anonymous temporary object called the *exception object*. The type of this exception object is used to transfer control to the nearest catch handler, which contains an exception declaration with a matching type. The C++ Standard, [except.handle], paragraph 16 [ISO/IEC 14882-2014], in part, states the following:

> *The variable declared by the* exception-declaration*, of type* cv *T or* cv *T&, is initialized from the exception object, of type E, as follows:*
> *— if T is a base class of E, the variable is copy-initialized from the corresponding base class subobject of the exception object;*
> *— otherwise, the variable is copy-initialized from the exception object.*

Because the variable declared by the *exception-declaration* is copy-initialized, it is possible to *slice* the exception object as part of the copy operation, losing valuable exception information and leading to incorrect error recovery. For more information about object slicing, see OOP51-CPP. Do not slice derived objects. Further, if the copy constructor of the exception object throws an exception, the copy initialization of the *exception-declaration* object results in undefined behavior. (See ERR60-CPP. Exception objects must be nothrow copy constructible for more information.)

Always catch exceptions by lvalue reference unless the type is a trivial type. For reference, the C++ Standard, [basic.types], paragraph 9 [ISO/IEC 14882-2014], defines trivial types as the following:

> *Arithmetic types, enumeration types, pointer types, pointer to member types, `std::nullptr_t`, and cv-qualified versions of these types are collectively called* scalar types*.... Scalar types, trivial class types, arrays of such types and cv-qualified versions of these types are collectively called* trivial types*.*

The C++ Standard, [class], paragraph 6, defines trivial class types as the following:

> *A* trivially copyable class *is a class that:*
> *— has no non-trivial copy constructors,*
> *— has no non-trivial move constructors,*
> *— has no non-trivial copy assignment operators,*
> *— has no non-trivial move assignment operators, and*
> *— has a trivial destructor.*
> *A* trivial class *is a class that has a default constructor, has no non-trivial default constructors, and is trivially copyable. [*Note*: In particular, a trivially copyable or trivial class does not have virtual functions or virtual base classes. —* end note*]*

Noncompliant Code Example

In this noncompliant code example, an object of type `S` is used to initialize the exception object that is later caught by an *exception-declaration* of type `std::exception`. The *exception-declaration* matches the exception object type, so the variable `E` is copy-initialized from the exception object, resulting in the exception object being sliced. Consequently, the output of this noncompliant code example is the implementation-defined value returned from calling `std::exception::what()` instead of `"My custom exception"`.

```
#include <exception>
#include <iostream>

struct S : std::exception {
  const char *what() const noexcept override {
    return "My custom exception";
  }
};

void f() {
  try {
    throw S();
  } catch (std::exception e) {
    std::cout << e.what() << std::endl;
  }
}
```

Compliant Solution

In this compliant solution, the variable declared by the *exception-declaration* is an lvalue reference. The call to `what()` results in executing `S::what()` instead of `std::exception::what()`.

```
#include <exception>
#include <iostream>

struct S : std::exception {
  const char *what() const noexcept override {
    return "My custom exception";
  }
};

void f() {
  try {
    throw S();
  } catch (std::exception &e) {
    std::cout << e.what() << std::endl;
  }
}
```

Risk Assessment

Object slicing can result in abnormal program execution. This generally is not a problem for exceptions, but it can lead to unexpected behavior depending on the assumptions made by the exception handler.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ERR61-CPP	Low	Unlikely	Low	**P3**	**L3**

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC++-ERR61**	
Clang	3.9	`cert-err61-cpp`	Checked by `clang-tidy`; also checks for VOID ERR09-CPP. Throw anonymous temporaries by default
SonarQube C/C++ Plugin	4.10	**S1044**	
LDRA tool suite	9.7.1	**455 S**	Fully implemented
Parasoft C/C++test	10.4.2	**CERT_CPP-ERR61-a** **CERT_CPP-ERR61-b**	A class type exception shall always be caught by reference Throw by value, catch by reference
Polyspace Bug Finder	R2019b	CERT C++: ERR61-CPP	Checks for exception object initialized by copy in catch statement (rule fully covered)
PRQA QA-C++	4.4	**4031**	
PVS-Studio	6.23	**V746**	

Related Vulnerabilities

Search for other vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

This rule is a subset of OOP51-CPP. Do not slice derived objects.

SEI CERT C++ Coding Standard	ERR60-CPP. Exception objects must be nothrow copy constructible

Bibliography

[ISO/IEC 14882-2014]	Subclause 3.9, "Types" Clause 9, "Classes" Subclause 15.1, "Throwing an Exception" Subclause 15.3, "Handling an Exception"

[MISRA 2008]	Rule 15-3-5