ERR60-CPP. Exception objects must be nothrow copy constructible

When an exception is thrown, the exception object operand of the `throw` expression is copied into a temporary object that is used to initialize the handler. The C++ Standard, [except.throw], paragraph 3 [ISO/IEC 14882-2014], in part, states the following:

> *Throwing an exception copy-initializes a temporary object, called the* exception object. *The temporary is an lvalue and is used to initialize the variable declared in the matching* handler.

If the copy constructor for the exception object type throws during the copy initialization, `std::terminate()` is called, which can result in undefined behavior. For more information on implicitly calling `std::terminate()`, see ERR50-CPP. Do not abruptly terminate the program.

The copy constructor for an object thrown as an exception must be declared `noexcept`, including any implicitly-defined copy constructors. Any function declared `noexcept` that terminates by throwing an exception violates ERR55-CPP. Honor exception specifications.

The C++ Standard allows the copy constructor to be elided when initializing the exception object to perform the initialization if a temporary is thrown. Many modern compiler implementations make use of both optimization techniques. However, the copy constructor for an exception object still must not throw an exception because compilers are not required to elide the copy constructor call in all situations, and common implementations of `std::exception_ptr` w ill call a copy constructor even if it can be elided from a `throw` expression.

Noncompliant Code Example

In this noncompliant code example, an exception of type `S` is thrown in `f()`. However, because `S` has a `std::string` data member, and the copy constructor for `std::string` is not declared `noexcept`, the implicitly-defined copy constructor for `S` is also not declared to be `noexcept`. In low-memory situations, the copy constructor for `std::string` may be unable to allocate sufficient memory to complete the copy operation, resulting in a `std::bad_alloc` exception being thrown.

```
#include <exception>
#include <string>

class S : public std::exception {
  std::string m;
public:
  S(const char *msg) : m(msg) {}

  const char *what() const noexcept override {
    return m.c_str();
  }
};

void g() {
  // If some condition doesn't hold...
  throw S("Condition did not hold");
}

void f() {
  try {
    g();
  } catch (S &s) {
    // Handle error
  }
}
```

Compliant Solution

This compliant solution assumes that the type of the exception object can inherit from `std::runtime_error`, or that type can be used directly. Unlike `std::string`, a `std::runtime_error` object is required to correctly handle an arbitrary-length error message that is exception safe and guarantees the copy constructor will not throw [ISO/IEC 14882-2014].

```
#include <stdexcept>
#include <type_traits>

struct S : std::runtime_error {
  S(const char *msg) : std::runtime_error(msg) {}
};

static_assert(std::is_nothrow_copy_constructible<S>::value,
              "S must be nothrow copy constructible");

void g() {
  // If some condition doesn't hold...
  throw S("Condition did not hold");
}

void f() {
  try {
    g();
  } catch (S &s) {
    // Handle error
  }
}
```

Compliant Solution

If the exception type cannot be modified to inherit from `std::runtime_error`, a data member of that type is a legitimate implementation strategy, as shown in this compliant solution.

```
#include <stdexcept>
#include <type_traits>

class S : public std::exception {
  std::runtime_error m;
public:
  S(const char *msg) : m(msg) {}

  const char *what() const noexcept override {
    return m.what();
  }
};

static_assert(std::is_nothrow_copy_constructible<S>::value,
              "S must be nothrow copy constructible");

void g() {
  // If some condition doesn't hold...
  throw S("Condition did not hold");
}

void f() {
  try {
    g();
  } catch (S &s) {
    // Handle error
  }
}
```

Risk Assessment

Allowing the application to abnormally terminate can lead to resources not being freed, closed, and so on. It is frequently a vector for denial-of-service attacks.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
ERR60-CPP	Low	Probable	Medium	**P4**	**L3**

Automated Detection

Tool	Version	Checker	Description
Clang	3.9	`cert-err60-cpp`	Checked by `clang-tidy`
Parasoft C /C++test	10.4.2	**CERT_CPP-ERR60-a** **CERT_CPP-ERR60-b**	Exception objects must be nothrow copy constructible An explicitly declared copy constructor for a class that inherits from 'std::exception' should have a non-throwing exception specification
PRQA QA-C++	4.4	**3508**	

Related Vulnerabilities

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

Related Guidelines

SEI CERT C++ Coding Standard	ERR50-CPP. Do not abruptly terminate the program ERR55-CPP. Honor exception specifications

Bibliography

[Hinnant 2015]	
[ISO/IEC 14882-2014]	Subclause 15.1, "Throwing an Exception" Subclause 18.8.1, "Class `exception`" Subclause 18.8.5, "Exception Propagation"