ERR51-CPP. Handle all exceptions

When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. The C++ Standard, [except.handle], paragraph 9 [ISO/IEC 14882-2014], states the following:

> *If no matching handler is found, the function `std::terminate()` is called; whether or not the stack is unwound before this call to s `td::terminate()` is implementation-defined.*

The default terminate handler called by `std::terminate()` calls `std::abort()`, which abnormally terminates the process. When `std::abort()` is called, or if the implementation does not unwind the stack prior to calling `std::terminate()`, destructors for objects may not be called and external resources can be left in an indeterminate state. Abnormal process termination is the typical vector for denial-of-service attacks. For more information on implicitly calling `std::terminate()`, see ERR50-CPP. Do not abruptly terminate the program.

All exceptions thrown by an application must be caught by a matching exception handler. Even if the exception cannot be gracefully recovered from, using the matching exception handler ensures that the stack will be properly unwound and provides an opportunity to gracefully manage external resources before terminating the process.

As per **ERR50-CPP-EX1**, a program that encounters an unrecoverable exception may explicitly catch the exception and terminate, but it may not allow the exception to remain uncaught. One possible solution to comply with this rule, as well as with ERR50-CPP, is for the `main()` function to catch all exceptions. While this does not generally allow the application to recover from the exception gracefully, it does allow the application to terminate in a controlled fashion.

Noncompliant Code Example

In this noncompliant code example, neither `f()` nor `main()` catch exceptions thrown by `throwing_func()`. Because no matching handler can be found for the exception thrown, `std::terminate()` is called.

```
void throwing_func() noexcept(false);

void f() {
  throwing_func();
}

int main() {
  f();
}
```

Compliant Solution

In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the `main()` function and allows for graceful management of external resources.

```
void throwing_func() noexcept(false);

void f() {
  throwing_func();
}

int main() {
  try {
    f();
  } catch (...) {
    // Handle error
  }
}
```

Noncompliant Code Example

In this noncompliant code example, the thread entry point function `thread_start()` does not catch exceptions thrown by `throwing_func()`. If the initial thread function exits because an exception is thrown, `std::terminate()` is called.

```
#include <thread>

void throwing_func() noexcept(false);

void thread_start() {
  throwing_func();
}

void f() {
  std::thread t(thread_start);
  t.join();
}
```

Compliant Solution

In this compliant solution, the `thread_start()` handles all exceptions and does not rethrow, allowing the thread to terminate normally.

```
#include <thread>

void throwing_func() noexcept(false);

void thread_start(void) {
  try {
    throwing_func();
  } catch (...) {
    // Handle error
  }
}

void f() {
  std::thread t(thread_start);
  t.join();
}
```

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
ERR51-CPP	Low	Probable	Medium	**P4**	**L3**

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC++-ERR51**	
LDRA tool suite	9.7.1	**527 S**	Partially implemented
Parasoft C/C++test	10.4.2	**CERT_CPP-ERR51-a** **CERT_CPP-ERR51-b**	Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point
Polyspace Bug Finder	R2019b	CERT C++: ERR51-CPP	Checks for unhandled exceptions (rule partially covered)
PRQA QA-C++	4.3	**4035, 4036, 4037**	

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 ERR50-CPP. Do not abruptly terminate the program.

MITRE CWE	CWE-754, Improper Check for Unusual or Exceptional Conditions

Bibliography

[ISO/IEC 14882-2014]	Subclause 15.1, "Throwing an Exception" Subclause 15.3, "Handling an Exception" Subclause 15.5.1, "The `std::terminate()` Function"
[MISRA 2008]	Rule 15-3-2 (Advisory) Rule 15-3-4 (Required)