UNCAUGHT_EXCEPT

Quality Checker

Overview

Supported Languages: C#, C/C++, CUDA, Objective-C/C++

`UNCAUGHT_EXCEPT` finds many cases in which an exception is thrown and never caught, or violates a function's exception specification. Usually, the result of such behavior is abnormal program termination.

The checker reports a defect if any of the following items occurs:

- An exception that is not allowed by the exception specification of a function is thrown.
- An exception is thrown from a root function. By default, a root function is defined as having no known callers, and its name matches the following regular expression:
  `((((^|_)m|M)ain)|(^MAIN))$`
  The preceding regular expression matches `main`, `WinMain`, `MAIN`. It does not match `DOMAIN`.

Note: By default, the checker ignores `bad_alloc` exceptions because `operator new` often throws this exception, and most programs are not affected by it. The `except_ignore` option to this checker and the `--handle-badalloc` option to `cov-analyze` override this default behavior.

Enablement

C#

Disabled by default: This checker is disabled by default. To enable it, you can use the `--enable` option to the `cov-analyze` command. For enablement/disablement details and options, see "Enabling and disabling checkers" in *Customizing Coverity*.

C++, CUDA, Objective-C++

Enabled by default: This checker is enabled by default. For enablement/disablement details and options, see "Enabling and disabling checkers" in *Customizing Coverity*.

Examples

This section provides one or more UNCAUGHT_EXCEPT examples.

```
// Example 1:
// Prototypical defect.
int main(){
  throw 7;
  return 0;
}
// Example 2:
// A simple defect resulting from a function call.
void fun() {
  throw 7;
}
int main(){
  fun();
  return 0;
}
// Example 3:
// An exception is thrown,
// violating the exception specification.
void fun() {
  throw 7;

}
void cannot_throw() throw() {
  fun();
}
// Example 4:
// An exception is thrown inside a try-catch block,
// but none of the catch statements has a matching
type.
class A {};
class B {};
class C {};

int main(){
  try {
    throw A();
  } catch (B b){
  } catch (C b){
  }
```

```
  return 0;
}
// Example 5:
// The exception is caught, but can be re-thrown.
class A {};

int main() {
  try {
    throw A();  //Will not be caught.
  } catch (...){
    cerr << "Error" << endl;
    throw;
  }
}
```

Options

This section describes one or more `UNCAUGHT_EXCEPT` options.

You can set specific checker option values by passing them with `--checker-option` to the `cov-analyze` command. For details, refer to the *Coverity 2023.12.2 Command Reference*.

- `UNCAUGHT_EXCEPT:except_ignore:<exception_class_identifier_pattern>` - This option excludes matching unqualified identifiers that escape a root function. The checker excludes an exception from the defect report if the pattern matches a class identifier for the exception. Default is unset.
  The checker treats the value to this option as an unanchored regular expression unless the value completely matches an exception class identifier. In the latter case, the checker only excludes full matches and does not exclude exceptions that partially match the value. You can specify this option multiple times.
  In the rare case that an exception is not an instance of a class, this option will not affect defect reporting on that exception.
  The checker runs this option after running `except_report`. Unlike `except_report`, this option does apply to exception-specification violations.
  If you use this option, the checker only excludes a `bad_alloc` exception if there is a matching value. Otherwise, it reports this exception.

- `UNCAUGHT_EXCEPT:except_report:<exception_class_identifier_pattern>` - This option finds matching unqualified identifiers that escape a root function. The checker includes an exception within the defect report if the pattern matches the class identifier for the exception. Default is unset.
  The checker treats the value to this option as an unanchored regular expression unless the value matches an exception class identifier completely. In the latter case, the checker only reports full matches and does not report exceptions that partially match the value. You can specify this option multiple times.
  You can use this option to force the checker to report `bad_alloc` exceptions. It has no effect on the reporting of exception-specification violations.
  This option is backwards compatible with pre-5.4 comma-separated string values.

- `UNCAUGHT_EXCEPT:follow_indirect_calls:<boolean>` - When this option is `true`, and either virtual function call tracking and/or function pointer tracking are enabled, UNCAUGHT_EXCEPT will follow such indirect calls for the purpose of propagating thrown exceptions. When false, exceptions are not considered to propagate across indirect calls, even when indirect call tracking is otherwise enabled. Defaults to `false`.

- `UNCAUGHT_EXCEPT:fun_ignore:<function_identifier_pattern>` - This option excludes an exception from a defect report if it results from a function that partially or fully matches the specified value. You specify function identifiers in the same was as you specify them for `fun_report`. Default is unset.
  This option does not apply to exception-specification violations.
  This option overrides the `fun_report` option.
  You can specify this option multiple times. The checker examines all matching values.

- `UNCAUGHT_EXCEPT:fun_report:<function_identifier_pattern>` - This option specifies a partially or completely matching function identifier. The checker treats the value to this option as an

unanchored regular expression. That is, a single identifier causes a full match, while a regular expression metacharacter yields a partial match. Default is unset.
If you specify `fun_report`, the checker treats:

- Any function that has an unqualified identifier (for example, `foo` in `bar::foo(int)`) that matches the <value> as a root function, and it reports any exceptions that escape from it as defects. The checker behaves in this manner regardless of whether other functions call the matching function or not.
- `main` and its variants (for example, `WinMain`, and `MAIN`) as entry points *only if* their function identifiers match one of the specified values.

- This option does not apply to exception-specification violations.
  You can specify this option multiple times. The checker examines all matching values.
- `UNCAUGHT_EXCEPT:report_all_fun:<boolean>` - When this option is set to `true`, it enables the reporting of exceptions for all functions that are not called by other functions. Defaults to `false`. This checker option is automatically set to `true` if the `--aggressiveness-level` option of the `cov-analyze` command is set to `high`.
- `UNCAUGHT_EXCEPT:report_exn_spec:<boolean>` - When this option is set to `false`, it disables reporting of exception-specification violations. Defaults to `true`.
- `UNCAUGHT_EXCEPT:report_thrown_pointers:<boolean>` - When this option is set to `true`, the checker reports an error when any pointer is thrown. Throwing by value is recommended, while throwing by pointer discouraged. Defaults to `false`.
  Example:

```
struct A { };

int main() {
  try {
      // The programmer actually wanted "throw
```

```
    A();"
•           throw new A();
•       } catch (A &a) {
•       } catch (...) {
•           // The exception is caught here, but was
    intended
•           // to be caught in the above block.
•       }
•     }
```

Events

This section describes one or more events produced by the `UNCAUGHT_EXCEPT` checker.

- Only one of the two following events is possible:
  - `exn_spec_violation` - Indicates that a function threw an exception that is not allowed by its exception specification.
  - `root_function` - Indicates that a root function does not catch an exception that could be thrown during its execution.
- Any number of the following events is possible:
  - `fun_call_w_exception` - Indicates that an exception is thrown by a function. It has a model link.
  - `fun_call_w_rethrow` - Indicates that a function has a `rethrow_outside_catch` event.
  - `rethrow` - Indicates that a `throw` statement re-throws an exception that is never caught.
  - `rethrow_outside_catch` - Indicates that `throw` statement occurred outside of function that contains a `try` statement.
  - `uncaught_exception` - Marks `throw` statements that produce an exception that will never be caught.