OOP56-CPP. Honor replacement handler requirements

The *handler* functions `new_handler`, `terminate_handler`, and `unexpected_handler` can be globally replaced by custom implementations, as specified by [handler.functions], paragraph 2, of the C++ Standard [ISO/IEC 14882-2014]. For instance, an application could set a custom termination handler by calling `std::set_terminate()`, and the custom termination handler may log the termination for later auditing. However, the C++ Standard, [res.on.functions], paragraph 1, states the following:

> *In certain cases (replacement functions, handler functions, operations on types used to instantiate standard library template components), the C++ standard library depends on components supplied by a C++ program. If these components do not meet their requirements, the Standard places no requirements on the implementation.*

Paragraph 2, in part, further states the following:

> *In particular, the effects are undefined in the following cases:*
> *— for handler functions, if the installed handler function does not implement the semantics of the applicable* Required behavior: *paragraph*

A replacement for any of the handler functions must meet the semantic requirements specified by the appropriate *Required behavior:* clause of the replaced function.

New Handler

The requirements for a replacement `new_handler` are specified by [new.handler], paragraph 2:

> *Required behavior: A `new_handler` shall perform one of the following:*
> *— make more storage available for allocation and then return;*
> *— throw an exception of type `bad_alloc` or a class derived from `bad_alloc`;*
> *— terminate execution of the program without returning to the caller;*

Terminate Handler

The requirements for a replacement `terminate_handler` are specified by [terminate.handler], paragraph 2:

> *Required behavior: A `terminate_handler` shall terminate execution of the program without returning to the caller.*

Unexpected Handler

The requirements for a replacement `unexpected_handler` are specified by [unexpected.handler], paragraph 2.

> *Required behavior: An `unexpected_handler` shall not return. See also 15.5.2.*

`unexpected_handler` is a deprecated feature of C++.

Noncompliant Code Example

In this noncompliant code example, a replacement `new_handler` is written to attempt to release salvageable resources when the dynamic memory manager runs out of memory. However, this example does not take into account the situation in which all salvageable resources have been recovered and there is still insufficient memory to satisfy the allocation request. Instead of terminating the replacement handler with an exception of type `std::bad_alloc` or terminating the execution of the program without returning to the caller, the replacement handler returns as normal. Under low memory conditions, an infinite loop will occur with the default implementation of `::operator new()`. Because such conditions are rare in practice, it is likely for this bug to go undiscovered under typical testing scenarios.

```
#include <new>

void custom_new_handler() {
  // Returns number of bytes freed.
  extern std::size_t reclaim_resources();
  reclaim_resources();
}

int main() {
  std::set_new_handler(custom_new_handler);

  // ...
}
```

Compliant Solution

In this compliant solution, `custom_new_handler()` uses the return value from `reclaim_resources()`. If it returns `0`, then there will be insufficient memory for `operator new` to succeed. Hence, an exception of type `std::bad_alloc` is thrown, meeting the requirements for the replacement handler.

```
#include <new>

void custom_new_handler() noexcept(false) {
  // Returns number of bytes freed.
  extern std::size_t reclaim_resources();
  if (0 == reclaim_resources()) {
    throw std::bad_alloc();
  }
}

int main() {
  std::set_new_handler(custom_new_handler);

  // ...
}
```

Risk Assessment

Failing to meet the required behavior for a replacement handler results in undefined behavior.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
OOP56-CPP	Low	Probable	High	**P2**	**L3**

Automated Detection

Tool	Version	Checker	Description
Parasoft C/C++test	10.4.2	**CERT_CPP-OOP56-a** **CERT_CPP-OOP56-b** **CERT_CPP-OOP56-c**	Properly define terminate handlers Properly define unexpected handlers Properly define new handlers

Related Vulnerabilities

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

Related Guidelines

SEI CERT C++ Coding Standard	MEM55-CPP. Honor replacement dynamic storage management requirements

Bibliography

[ISO/IEC 14882-2014]	Subclause 17.6.4.8, "Other Functions" Subclause 18.6.2.3, "Type `new_handler`" Subclause 18.8.3.1, "Type `terminate_handler`" Subclause D.11.1, "Type `unexpected_handler`"