MEM55-CPP. Honor replacement dynamic storage management requirements

Dynamic memory allocation and deallocation functions can be globally replaced by custom implementations, as specified by [replacement.functions], paragraph 2, of the C++ Standard [ISO/IEC 14882-2014]. For instance, a user may profile the dynamic memory usage of an application and decide that the default allocator is not optimal for their usage pattern, and a different allocation strategy may be a marked improvement. 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 further, in part, states the following:

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

A replacement for any of the dynamic memory allocation or deallocation functions must meet the semantic requirements specified by the appropriate *Required behavior:* clause of the replaced function.

Noncompliant Code Example

In this noncompliant code example, the global `operator new(std::size_t)` function is replaced by a custom implementation. However, the custom implementation fails to honor the behavior required by the function it replaces, as per the C++ Standard, [new.delete.single], paragraph 3. Specifically, if the custom allocator fails to allocate the requested amount of memory, the replacement function returns a null pointer instead of throwing an exception of type `std::bad_alloc`. By returning a null pointer instead of throwing, functions relying on the required behavior of `operator new(std::size_t)` to throw on memory allocations may instead attempt to dereference a null pointer. See EXP34-C. Do not dereference null pointers for more information.

```
#include <new>

void *operator new(std::size_t size) {
  extern void *alloc_mem(std::size_t); // Implemented elsewhere; may return nullptr
  return alloc_mem(size);
}

void operator delete(void *ptr) noexcept; // Defined elsewhere
void operator delete(void *ptr, std::size_t) noexcept; // Defined elsewhere
```

The declarations of the replacement `operator delete()` functions indicate that this noncompliant code example still complies with DCL54-CPP. Overload allocation and deallocation functions as a pair in the same scope.

Compliant Solution

This compliant solution implements the required behavior for the replaced global allocator function by properly throwing a `std::bad_alloc` exception when the allocation fails.

```
#include <new>

void *operator new(std::size_t size) {
  extern void *alloc_mem(std::size_t); // Implemented elsewhere; may return nullptr
  if (void *ret = alloc_mem(size)) {
    return ret;
  }
  throw std::bad_alloc();
}

void operator delete(void *ptr) noexcept; // Defined elsewhere
void operator delete(void *ptr, std::size_t) noexcept; // Defined elsewhere
```

Risk Assessment

Failing to meet the stated requirements for a replaceable dynamic storage function leads to undefined behavior. The severity of risk depends heavily on the caller of the allocation functions, but in some situations, dereferencing a null pointer can lead to the execution of arbitrary code [Jack 2007, van Sprundel 2006]. The indicated severity is for this more severe case.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
MEM55-CPP	High	Likely	Medium	P18	L1

Automated Detection

Tool	Version	Checker	Description
Parasoft C/C++test	10.4.2	**CERT_CPP-MEM55-a**	The user defined 'new' operator should throw the 'std::bad_alloc' exception when the allocation fails

Related Vulnerabilities

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

Related Guidelines

SEI CERT C++ Coding Standard	DCL54-CPP. Overload allocation and deallocation functions as a pair in the same scope OOP56-CPP. Honor replacement handler requirements
SEI CERT C Coding Standard	EXP34-C. Do not dereference null pointers

Bibliography

[ISO/IEC 14882-2014]	Subclause 17.6.4.8, "Other Functions" Subclause 18.6.1, "Storage Allocation and Deallocation"
[Jack 2007]	
[van Sprundel 2006]	