ERR03-C. Use runtime-constraint handlers when calling the bounds-checking interfaces

Most functions defined by the C Standard, Annex K Bounds-checking interfaces, include, as part of their specification, a list of runtime constraints, violations of which can be consistently handled at runtime. Library implementations must verify that the runtime constraints for a function are not violated by the program. If a runtime constraint is violated, the runtime-constraint handler currently registered with `set_constraint_handler_s()` is called.

Annex K, subclause K.3.6.1.1, of the C Standard [ISO/IEC 9899:2011] states:

> *When the handler is called, it is passed the following arguments in the following order:*
>
> 1. *A pointer to a character string describing the runtime-constraint violation.*
> 2. *A null pointer or a pointer to an implementation-defined object.*
> 3. *If the function calling the handler has a return type declared as `errno_t`, the return value of the function is passed. Otherwise, a positive value of type `errno_t` is passed.*
>
> *The implementation has a default constraint handler that is used if no calls to the `set_constraint_handler_s()` function have been made or the handler argument to `set_constraint_handler_s()` is a null pointer. The behavior of the default handler is implementation-defined, and it may cause the program to exit or abort.*

And subclause K.3.1.4 states:

> *The runtime-constraint handler might not return. If the handler does return, the library function whose runtime-constraint was violated shall return some indication of failure as given by the returns section in the function's specification.*

These runtime-constraint handlers mitigate some of the potential insecurity caused by in-band error indicators. (See ERR02-C. Avoid in-band error indicators.)

Noncompliant Code Example (C11 Annex K)

In this noncompliant code example, the `strcpy_s()` function is called, but no runtime-constraint handler has been explicitly registered. As a result, the implementation-defined default handler is called on a runtime error.

```
errno_t function(char *dst1, size_t size){
  char src1[100] = "hello";

  if (strcpy_s(dst1, size, src1) != 0) {
    return -1;
  }
  /* ... */
  return 0;
}
```

The result is inconsistent behavior across implementations and possible termination of the program instead of a graceful exit. The implementation-defined default handler performs a default action consistent with a particular implementation. However, this may not be the desired action, and because the behavior is implementation-defined, it is not guaranteed to be the same on all implementations.

It is therefore prudent to explicitly install a runtime-constraint handler to ensure consistent behavior across implementations.

Compliant Solution (C11 Annex K)

This compliant solution explicitly installs a runtime-constraint handler by invoking the `set_constraint_handler_s()` function. It would typically be performed during system initialization and before any functions that used the mechanism were invoked.

```
constraint_handler_t handle_errors(void) {
  /* Handle runtime-constraint error */
}

/* ... */

set_constraint_handler_s(handle_errors);

/* ... */

/* Returns zero on success */
errno_t function(char *dst1, size_t size){
  char src1[100] = "hello";

  if (strcpy_s(dst1, size, src1) != 0) {
    return -1;
  }
  /* ... */
  return 0;
}
```

Compliant Solution (Visual Studio 2008 and later)

Although the C11 Annex K functions were created by Microsoft, Microsoft Visual Studio does not support the same interface defined by the technical report for installing runtime-constraint handlers. Visual Studio calls these functions *invalid parameter handlers*, and they are installed by calling the `_set_invalid_parameter_handler()` function. The signature of the handler is also significantly different [MSDN].

```
_invalid_parameter_handler handle_errors(
   const wchar_t* expression,
   const wchar_t* function,
   const wchar_t* file,
   unsigned int line,
   uintptr_t pReserved
) {
  /* Handle invalid parameter */
}

/* ... */

_set_invalid_parameter_handler(handle_errors)

/* ... */

errno_t function(char *dst1, size_t size) {
  char src1[100] = "hello";

  if (strcpy_s(dst1, size, src1) != 0) {
    return -1;
  }
  /* ... */
  return 0;
}
```

Risk Assessment

C11 Annex K indicates that if no constraint handler is set, a default one executes when errors arise. The default handler is implementation-defined and "may cause the program to exit or abort" [ISO/IEC 9899:2011]. It is important to understand the behavior of the default handler for all implementations being used and replace it if the behavior is inappropriate for the application.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
ERR03-C	Low	Unlikely	Medium	P2	L3

Related Vulnerabilities

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

Related Guidelines

SEI CERT C++ Coding Standard	VOID ERR03-CPP. Use runtime-constraint handlers when calling functions defined by TR24731-1

Bibliography

[ISO/IEC 9899:2011]	Subclause K.3.1.4, "Runtime-Constraint Violations" Subclause K.3.6.1, "Runtime-Constraint Handling"
[MSDN]	"Parameter Validation"