Relation to the CERT C Coding Standard

The C++ Standard, [intro.scope], paragraph 2 [ISO/IEC 14882-2014] states the following:

> *C++ is a general purpose programming language based on the C programming language as described in ISO/IEC 9899:1999 Programming languages—C (hereinafter referred to as the C standard). In addition to the facilities provided by C, C++ provides additional data types, classes, templates, exceptions, namespaces, operator overloading, function name overloading, references, free store management operators, and additional library facilities.*

Because C++ is based on the C programming language, there is considerable overlap between the guidelines specified by the SEI CERT C Coding Standard and those specified by this coding standard. To reduce the amount of duplicated information, this coding standard focuses on the parts of the C++ programming language that are not wholly covered by the CERT C Coding Standard. Because of the increased focus on types in C++, some rules in C are extended by the CERT C++ Secure Coding Standard.

Rules from the CERT C Coding Standard that apply to the CERT C++ Coding Standard are described in each related chapter of the C++ standard. The POSIX (POS) and Microsoft Windows (WIN) rules from the CERT C Coding Standard have not been reviewed for applicability to code written in C++ for those platforms.

Recommendations from the CERT C Coding Standard have not been reviewed for applicability to code written in C++.

The following rules from the CERT C Coding Standard have been reviewed and **do not** apply to the CERT C++ Secure Coding Standard:

- ARR32-C. Ensure size arguments for variable length arrays are in a valid range
- ARR36-C. Do not subtract or compare two pointers that do not refer to the same array
- CON30-C. Clean up thread-specific storage
- CON31-C. Do not destroy a mutex while it is locked
- CON32-C. Prevent data races when accessing bit-fields from multiple threads
- CON34-C. Declare objects shared between threads with appropriate storage durations
- CON35-C. Avoid deadlock by locking in a predefined order
- CON36-C. Wrap functions that can spuriously wake up in a loop
- CON38-C. Preserve thread safety and liveness when using condition variables
- CON39-C. Do not join or detach a thread that was previously joined or detached
- DCL31-C. Declare identifiers before using them
- DCL36-C. Do not declare an identifier with conflicting linkage classifications
- DCL37-C. Do not declare or define a reserved identifier
- DCL38-C. Use the correct syntax when declaring a flexible array member
- DCL41-C. Do not declare variables inside a switch statement before the first case label
- EXP30-C. Do not depend on the order of evaluation for side effects
- EXP32-C. Do not access a volatile object through a nonvolatile reference
- EXP33-C. Do not read uninitialized memory
- EXP40-C. Do not modify constant objects
- EXP43-C. Avoid undefined behavior when using restrict-qualified pointers
- EXP44-C. Do not rely on side effects in operands to sizeof, _Alignof, or _Generic
- MEM33-C. Allocate and copy structures containing a flexible array member dynamically
- SIG30-C. Call only asynchronous-safe functions within signal handlers