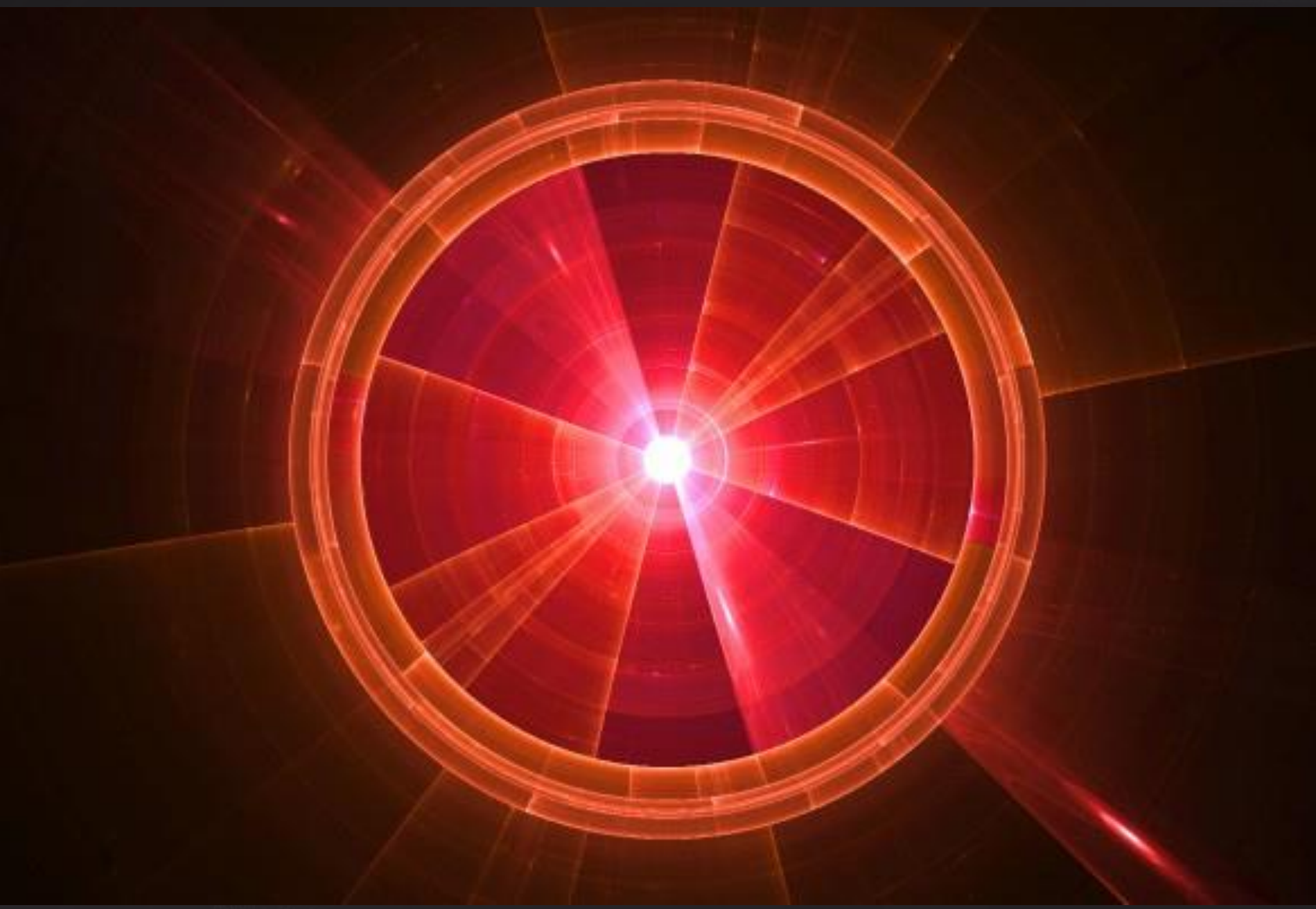
GRAMMATECH

SEI CERT-C++ RULES AND RECOMMENDATIONS

MAPPED TO CODESONAR® 7.1 WARNING CLASSES



TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

INTRODUCTION

The SEI CERT C++ Coding Standard (CERT-C++) provides rules and recommendations for securecoding in the C++ programming language. The goal of these rules and recommendations is to develop safe, reliable, and secure systems, for example by eliminating undefined behaviors that can lead to undefined program behaviors and exploitable vulnerabilities. Conformance to the coding rules defined in this standard is necessary (but not sufficient) to ensure the safety, reliability, and security of software systems developed in the C++ programming language.

CodeSonar 7.1 includes a large number of warning classes that support checking for theCERT-C++ rules and recommendations. Every CodeSonar warning report includes the identifiersof any CERT-C++ rules and recommendations that are closely mapped to the warning's class. (The close mapping for a warning class is the set of categories—including CERT-C++ rules and recommendations—that most closely match the class, if any).

You can configure CodeSonar to enable and disable warning classes mapped to specific CERT-C++rules and recommendations, or use build presets to enable all warning classes that are closely mapped to any CERT-C++ rules and recommendations. In addition, you can use the CodeSonar search function to find warnings related to specific CERT-C++ rules or recommendations, or to any CERT-C++ rule or recommendation.

For more information on the SEI CERT C++ Coding Standard:

https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=8804668

The remainder of this document comprises two tables:

- A table showing the close mapping between CodeSonar warning classes and the SEI CERTC++ Coding Standard.
- A table showing the broad mapping between CodeSonar warning classes and the SEI CERTC++ Coding Standard. The broad CERT-C++ mapping for a CodeSonar warning class includes the close CERT-C++ mapping for the class, plus any other CERT-C++ rules and recommendations that are related to the class in a meaningful way, but not eligible for the close mapping.

GrammaTech is a leading global provider of application testing (AST) solutions used by the world's most security conscious organizations to detect, measure, analyze and resolve vulnerabilities for software they develop or use. The company is also a trusted cybersecurity and artificial intelligence research partner for the nation's civil, defense, and intelligence agencies.

SEI CERT C++ CODING STANDARD CLOSE MAPPING (CODESONAR V7.1)

The following table contains CodeSonar warning classes that are closely mapped to CERT-C++ rules and recommendations.

Rule	Rule Name	Supported
CERT-CPP:CON50-CPP	Do not destroy a mutex while it is locked	Yes
CERT-CPP:CON51-CPP	Ensure actively held locks are released on exceptional conditions	Yes
CERT-CPP:CON52-CPP	Prevent data races when accessing bit-fields from multiple threads	Yes
CERT-CPP:CON53-CPP	Avoid deadlock by locking in a predefined order	Yes
CERT-CPP:CON54-CPP	Wrap functions that can spuriously wake up in a loop	Yes
CERT-CPP:CON55-CPP	Preserve thread safety and liveness when using condition variables	Yes
CERT-CPP:CON56-CPP	Do not speculatively lock a non-recursive mutex that is already owned by the calling thread	Yes
CERT-CPP:CTR50-CPP	Guarantee that container indices and iterators are within the valid range	Yes
CERT-CPP:CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container	Yes
CERT-CPP:CTR52-CPP	Guarantee that library functions do not overflow	Yes
CERT-CPP:CTR53-CPP	Use valid iterator ranges	Yes
CERT-CPP:CTR54-CPP	Do not subtract iterators that do not refer to the same container	Yes
CERT-CPP:CTR55-CPP	Do not use an additive operator on an iterator if the result would overflow	No
CERT-CPP:CTR56-CPP	Do not use pointer arithmetic on polymorphic objects	Yes
CERT-CPP:CTR57-CPP	Provide a valid ordering predicate	No
CERT-CPP:CTR58-CPP	Predicate function objects should not be mutable	No
CERT-CPP:DCL50-CPP	Do not define a C-style variadic function	Yes
CERT-CPP:DCL51-CPP	Do not declare or define a reserved identifier	Yes
CERT-CPP:DCL52-CPP	Never qualify a reference type with const or volatile	No
CERT-CPP:DCL53-CPP	Do not write syntactically ambiguous declarations	Yes
CERT-CPP:DCL54-CPP	Overload allocation and deallocation functions as a pair in the same scope	No
CERT-CPP:DCL55-CPP	Avoid information leakage when passing a class object across a trust boundary	Yes
CERT-CPP:DCL56-CPP	Avoid cycles during initialization of static objects	Yes
CERT-CPP:DCL57-CPP	Do not let exceptions escape from destructors or deallocation functions	Yes
CERT-CPP:DCL58-CPP	Do not modify the standard namespaces	Yes
CERT-CPP:DCL59-CPP	Do not define an unnamed namespace in a header file	Yes
CERT-CPP:DCL60-CPP	Obey the one-definition rule	Yes
CERT-CPP:ERR50-CPP	Do not abruptly terminate the program	Yes
CERT-CPP:ERR51-CPP	Handle all exceptions	Yes
CERT-CPP:ERR52-CPP	Do not use setjmp() or longjmp()	Yes
CERT-CPP:ERR53-CPP	Do not reference base classes or class data members in a constructor or destructor function-try-block handler	No
CERT-CPP:ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived	Yes
CERT-CPP:ERR55-CPP	Honor exception specifications	Yes
CERT-CPP:ERR56-CPP	Guarantee exception safety	No
CERT-CPP:ERR57-CPP	Do not leak resources when handling exceptions	Yes
CERT-CPP:ERR58-CPP	Handle all exceptions thrown before main() begins executing	Yes
CERT-CPP:ERR59-CPP	Do not throw an exception across execution boundaries	No
CERT-CPP:ERR60-CPP	Exception objects must be nothrow copy constructible	No

CERT-CPP:ERR61-CPP	Catch exceptions by lvalue reference	Yes
CERT-CPP:ERR62-CPP	Detect errors when converting a string to a number	Yes
CERT-CPP:EXP50-CPP	Do not depend on the order of evaluation for side effects	Yes
CERT-CPP:EXP51-CPP	Do not delete an array through a pointer of the incorrect type	Yes
CERT-CPP:EXP52-CPP	Do not rely on side effects in unevaluated operands	Yes
CERT-CPP:EXP53-CPP	Do not read uninitialized memory	Yes
CERT-CPP:EXP54-CPP	Do not access an object outside of its lifetime	Yes
CERT-CPP:EXP55-CPP	Do not access a cv-qualified object through a cv-unqualified type	No
CERT-CPP:EXP56-CPP	Do not call a function with a mismatched language linkage	No
CERT-CPP:EXP57-CPP	Do not cast or delete pointers to incomplete classes	Yes
CERT-CPP:EXP58-CPP	Pass an object of the correct type to va_start	Yes
CERT-CPP:EXP59-CPP	Use offsetof() on valid types and members	Yes
CERT-CPP:EXP60-CPP	Do not pass a nonstandard-layout type object across execution boundaries	No
CERT-CPP:EXP61-CPP	A lambda object must not outlive any of its reference captured objects	No
CERT-CPP:EXP62-CPP	Do not access the bits of an object representation that are not part of the object's value representation	Yes
CERT-CPP:EXP63-CPP	Do not rely on the value of a moved-from object	No
CERT-CPP:FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call	Yes
CERT-CPP:FIO51-CPP	Close files when they are no longer needed	Yes
CERT-CPP:INT50-CPP	Do not cast to an out-of-range enumeration value	Yes
CERT-CPP:MEM50-CPP	Do not access freed memory	Yes
CERT-CPP:MEM51-CPP	Properly deallocate dynamically allocated resources	Yes
CERT-CPP:MEM52-CPP	Detect and handle memory allocation errors	No
CERT-CPP:MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	No
CERT-CPP:MEM54-CPP	Provide placement new with properly aligned pointers to sufficient storage capacity	Yes
CERT-CPP:MEM55-CPP	Honor replacement dynamic storage management requirements	No
CERT-CPP:MEM56-CPP	Do not store an already-owned pointer value in an unrelated smart pointer	No
CERT-CPP:MEM57-CPP	Avoid using default operator new for over-aligned types	No
CERT-CPP:MSC50-CPP	Do not use std::rand() for generating pseudorandom numbers	Yes
CERT-CPP:MSC51-CPP	Ensure your random number generator is properly seeded	Yes
CERT-CPP:MSC52-CPP	Value-returning functions must return a value from all exit paths	Yes
CERT-CPP:MSC53-CPP	Do not return from a function declared [[noreturn]]	Yes
CERT-CPP:MSC54-CPP	A signal handler must be a plain old function	No
CERT-CPP:OOP50-CPP	Do not invoke virtual functions from constructors or destructors	Yes
CERT-CPP:OOP51-CPP	Do not slice derived objects	Yes
CERT-CPP:OOP52-CPP	Do not delete a polymorphic object without a virtual destructor	Yes
CERT-CPP:OOP53-CPP	Write constructor member initializers in the canonical order	Yes
CERT-CPP:OOP54-CPP	Gracefully handle self-copy assignment	Yes
CERT-CPP:OOP55-CPP	Do not use pointer-to-member operators to access nonexistent members	Yes
CERT-CPP:OOP56-CPP	Honor replacement handler requirements	No
CERT-CPP:OOP57-CPP	Prefer special member functions and overloaded operators to C Standard Library functions	Yes
CERT-CPP:OOP58-CPP	Copy operations must not mutate the source object	Yes
CERT-CPP:STR50-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator	Yes
CERT-CPP:STR51-CPP	Do not attempt to create a std::string from a null pointer	Yes
CERT-CPP:STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string	Yes
CERT-CPP:STR53-CPP	Range check element access	No

SEI CERT C++ CODING STANDARD BROAD MAPPING (CODESONAR V7.1)

The following table contains CodeSonar warning classes that are broadly mapped to CERT-C++ rules and recommendations.

Rule	Rule Name	Supported
CERT-CPP:CON50-CPP	Do not destroy a mutex while it is locked	Yes
CERT-CPP:CON51-CPP	Ensure actively held locks are released on exceptional conditions	Yes
CERT-CPP:CON52-CPP	Prevent data races when accessing bit-fields from multiple threads	Yes
CERT-CPP:CON53-CPP	Avoid deadlock by locking in a predefined order	Yes
CERT-CPP:CON54-CPP	Wrap functions that can spuriously wake up in a loop	Yes
CERT-CPP:CON55-CPP	Preserve thread safety and liveness when using condition variables	Yes
CERT-CPP:CON56-CPP	Do not speculatively lock a non-recursive mutex that is already owned by the calling thread	Yes
CERT-CPP:CTR50-CPP	Guarantee that container indices and iterators are within the valid range	Yes
CERT-CPP:CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container	Yes
CERT-CPP:CTR52-CPP	Guarantee that library functions do not overflow	Yes
CERT-CPP:CTR53-CPP	Use valid iterator ranges	Yes
CERT-CPP:CTR54-CPP	Do not subtract iterators that do not refer to the same container	Yes
CERT-CPP:CTR55-CPP	Do not use an additive operator on an iterator if the result would overflow	Yes
CERT-CPP:CTR56-CPP	Do not use pointer arithmetic on polymorphic objects	Yes
CERT-CPP:CTR57-CPP	Provide a valid ordering predicate	No
CERT-CPP:CTR58-CPP	Predicate function objects should not be mutable	No
CERT-CPP:DCL50-CPP	Do not define a C-style variadic function	Yes
CERT-CPP:DCL51-CPP	Do not declare or define a reserved identifier	Yes
CERT-CPP:DCL52-CPP	Never qualify a reference type with const or volatile	Yes
CERT-CPP:DCL53-CPP	Do not write syntactically ambiguous declarations	Yes
CERT-CPP:DCL54-CPP	Overload allocation and deallocation functions as a pair in the same scope	Yes
CERT-CPP:DCL55-CPP	Avoid information leakage when passing a class object across a trust boundary	Yes
CERT-CPP:DCL56-CPP	Avoid cycles during initialization of static objects	Yes
CERT-CPP:DCL57-CPP	Do not let exceptions escape from destructors or deallocation functions	Yes
CERT-CPP:DCL58-CPP	Do not modify the standard namespaces	Yes
CERT-CPP:DCL59-CPP	Do not define an unnamed namespace in a header file	Yes
CERT-CPP:DCL60-CPP	Obey the one-definition rule	Yes
CERT-CPP:ERR50-CPP	Do not abruptly terminate the program	Yes
CERT-CPP:ERR51-CPP	Handle all exceptions	Yes
CERT-CPP:ERR52-CPP	Do not use setjmp() or longjmp()	Yes
CERT-CPP:ERR53-CPP	Do not reference base classes or class data members in a constructor or destructor function-try-block handler	No
CERT-CPP:ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived	Yes
CERT-CPP:ERR55-CPP	Honor exception specifications	Yes
CERT-CPP:ERR56-CPP	Guarantee exception safety	Yes
CERT-CPP:ERR57-CPP	Do not leak resources when handling exceptions	Yes
CERT-CPP:ERR58-CPP	Handle all exceptions thrown before main() begins executing	Yes
CERT-CPP:ERR59-CPP	Do not throw an exception across execution boundaries	No
CERT-CPP:ERR60-CPP	Exception objects must be nothrow copy constructible	No
CERT-CPP:ERR61-CPP	Catch exceptions by lvalue reference	Yes
CERT-CPP:ERR62-CPP	Detect errors when converting a string to a number	Yes
CERT-CPP:EXP50-CPP	Do not depend on the order of evaluation for side effects	Yes

CERT-CPP:EXP51-CPP	Do not delete an array through a pointer of the incorrect type	Yes
CERT-CPP:EXP52-CPP	Do not rely on side effects in unevaluated operands	Yes
CERT-CPP:EXP53-CPP	Do not read uninitialized memory	Yes
CERT-CPP:EXP54-CPP	Do not access an object outside of its lifetime	Yes
CERT-CPP:EXP55-CPP	Do not access a cv-qualified object through a cv-unqualified type	Yes
CERT-CPP:EXP56-CPP	Do not call a function with a mismatched language linkage	No
CERT-CPP:EXP57-CPP	Do not cast or delete pointers to incomplete classes	Yes
CERT-CPP:EXP58-CPP	Pass an object of the correct type to va_start	Yes
CERT-CPP:EXP59-CPP	Use offsetof() on valid types and members	Yes
CERT-CPP:EXP60-CPP	Do not pass a nonstandard-layout type object across execution boundaries	No
CERT-CPP:EXP61-CPP	A lambda object must not outlive any of its reference captured objects	No
CERT-CPP:EXP62-CPP	Do not access the bits of an object representation that are not part of the object's value representation	Yes
CERT-CPP:EXP63-CPP	Do not rely on the value of a moved-from object	Yes
CERT-CPP:FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call	Yes
CERT-CPP:FIO51-CPP	Close files when they are no longer needed	Yes
CERT-CPP:INT50-CPP	Do not cast to an out-of-range enumeration value	Yes
CERT-CPP:MEM50-CPP	Do not access freed memory	Yes
CERT-CPP:MEM51-CPP	Properly deallocate dynamically allocated resources	Yes
CERT-CPP:MEM52-CPP	Detect and handle memory allocation errors	Yes
CERT-CPP:MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	Yes
CERT-CPP:MEM54-CPP	Provide placement new with properly aligned pointers to sufficient storage capacity	Yes
CERT-CPP:MEM55-CPP	Honor replacement dynamic storage management requirements	No
CERT-CPP:MEM56-CPP	Do not store an already-owned pointer value in an unrelated smart pointer	Yes
CERT-CPP:MEM57-CPP	Avoid using default operator new for over-aligned types	No
CERT-CPP:MSC50-CPP	Do not use std::rand() for generating pseudorandom numbers	Yes
CERT-CPP:MSC51-CPP	Ensure your random number generator is properly seeded	Yes
CERT-CPP:MSC52-CPP	Value-returning functions must return a value from all exit paths	Yes
CERT-CPP:MSC53-CPP	Do not return from a function declared [[noreturn]]	Yes
CERT-CPP:MSC54-CPP	A signal handler must be a plain old function	No
CERT-CPP:OOP50-CPP	Do not invoke virtual functions from constructors or destructors	Yes
CERT-CPP:OOP51-CPP	Do not slice derived objects	Yes
CERT-CPP:OOP52-CPP	Do not delete a polymorphic object without a virtual destructor	Yes
CERT-CPP:OOP53-CPP	Write constructor member initializers in the canonical order	Yes
CERT-CPP:OOP54-CPP	Gracefully handle self-copy assignment	Yes
CERT-CPP:OOP55-CPP	Do not use pointer-to-member operators to access nonexistent members	Yes
CERT-CPP:OOP56-CPP	Honor replacement handler requirements	No
CERT-CPP:OOP57-CPP	Prefer special member functions and overloaded operators to C Standard Library functions	Yes
CERT-CPP:OOP58-CPP	Copy operations must not mutate the source object	Yes
CERT-CPP:STR50-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator	Yes
CERT-CPP:STR51-CPP	Do not attempt to create a std::string from a null pointer	Yes
CERT-CPP:STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string	Yes
CERT-CPP:STR53-CPP	Range check element access	Yes