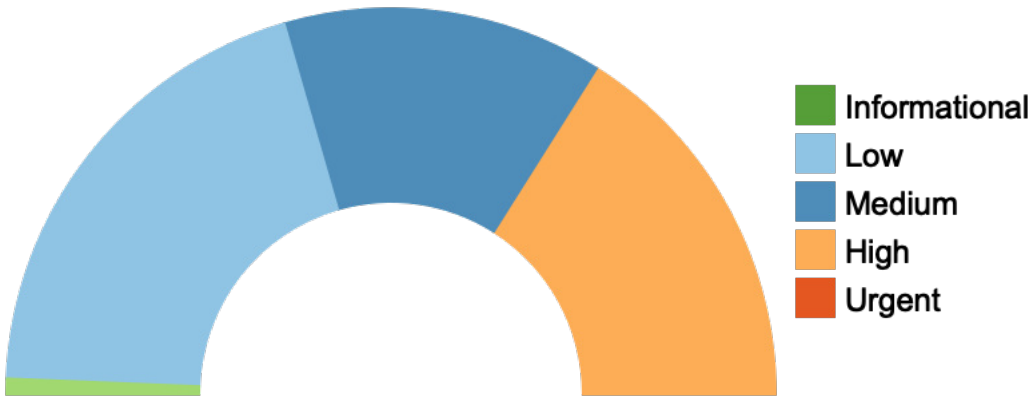
This standard provides rules for secure coding in the C++ programming language.

The rules and recommendations in this standard are a work in progress and reflect the current thinking of the secure coding community. As rules and recommendations mature, they are published in report or book form as official releases. These releases are issued as dictated by the needs and interests of the secure software development community.

The CERT C++ Coding Standard does not currently expose any recommendations; all C++ recommendations have been removed (moved to The Void section) due to quality concerns pending further review and development.

The list of rules and recommendations in this tool were last updated on 2023/05/23.

Checks by Severity



Checks

Check ID	Check Name	Supported	Severity
CON50-CPP	Do not destroy a mutex while it is locked	Yes	Medium
CON51-CPP	Ensure actively held locks are released on exceptional conditions	Yes	Low
CON52-CPP	Prevent data races when accessing bit-fields from multiple threads	Yes	Medium
CON53-CPP	Avoid deadlock by locking in a predefined order	No	Low
CON54-CPP	Wrap functions that can spuriously wake up in a loop	Yes	Medium
CON55-CPP	Preserve thread safety and liveness when using condition variables	Yes	Low
CON56-CPP	Do not speculatively lock a non-recursive	Yes	Low

	mutex that is already owned by the calling thread		
CTR50-CPP	Guarantee that container indices and iterators are within the valid range	Yes	High
CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container	Yes	High
CTR52-CPP	Guarantee that library functions do not overflow	Yes	High
CTR53-CPP	Use valid iterator ranges	Yes	High
CTR54-CPP	Do not subtract iterators that do not refer to the same container	Yes	Medium
CTR55-CPP	Do not use an additive operator on an iterator if the result would overflow	Yes	High
CTR56-CPP	Do not use pointer arithmetic on polymorphic objects	Yes	High
CTR57-CPP	Provide a valid ordering predicate	Yes	Low
CTR58-CPP	Predicate function objects should not be mutable	Yes	Low
DCL50-CPP	Do not define a C-style variadic function	Yes	High
DCL51-CPP	Do not declare or define a reserved identifier	No	
DCL52-CPP	Never qualify a reference type with const or volatile	Yes	Low
DCL53-CPP	Do not write syntactically ambiguous declarations	Yes	Low
DCL54-CPP	Overload allocation and deallocation functions as a pair in the same scope	Yes	Low
DCL55-CPP	Avoid information leakage when passing a class object across a trust boundary	No	Low
DCL56-CPP	Avoid cycles during initialization of static objects	Yes	Low
DCL57-CPP	Do not let exceptions escape from destructors or deallocation functions	Yes	Low
DCL58-CPP	Do not modify the standard namespaces	Yes	High
DCL59-CPP	Do not define an unnamed namespace in a header file	Yes	Medium
DCL60-CPP	Obey the one-definition rule	Yes	High
ERR50-CPP	Do not abruptly terminate the program	Yes	Low
ERR51-CPP	Handle all exceptions	Yes	Low
ERR52-CPP	Do not use setjmp() or longjmp()	Yes	Low
ERR53-CPP	Do not reference base classes or class data	Yes	Low

	members in a constructor or destructor function-try-block handler		
ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived	Yes	Medium
ERR55-CPP	Honor exception specifications	Yes	Low
ERR56-CPP	Guarantee exception safety	No	
ERR57-CPP	Do not leak resources when handling exceptions	Yes	Low
ERR58-CPP	Handle all exceptions thrown before main() begins executing	Yes	Low
ERR59-CPP	Do not throw an exception across execution boundaries	Yes	High
ERR60-CPP	Exception objects must be nothrow copy constructible	Yes	Low
ERR61-CPP	Catch exceptions by lvalue reference	Yes	Low
ERR62-CPP	Detect errors when converting a string to a number	Yes	Medium
EXP50-CPP	Do not depend on the order of evaluation for side effects	Yes	Medium
EXP51-CPP	Do not delete an array through a pointer of the incorrect type	Yes	Low
EXP52-CPP	Do not rely on side effects in unevaluated operands	Yes	Low
EXP53-CPP	Do not read uninitialized memory	Yes	High
EXP54-CPP	Do not access an object outside of its lifetime	Yes	High
EXP55-CPP	Do not access a cv-qualified object through a cv-unqualified type	Yes	Medium
EXP56-CPP	Do not call a function with a mismatched language linkage	No	Low
EXP57-CPP	Do not cast or delete pointers to incomplete classes	Yes	Medium
EXP58-CPP	Pass an object of the correct type to va_start	Yes	Medium
EXP59-CPP	Use offsetof() on valid types and members	Yes	Medium
EXP60-CPP	Do not pass a nonstandard-layout type object across execution boundaries	No	
EXP61-CPP	A lambda object must not outlive any of its reference captured objects	Yes	High
EXP62-CPP	Do not access the bits of an object	Yes	High

	representation that are not part of the object's value representation		
EXP63-CPP	Do not rely on the value of a moved-from object	Yes	Medium
FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call	Yes	Low
FIO51-CPP	Close files when they are no longer needed	Yes	Medium
INT50-CPP	Do not cast to an out-of-range enumeration value	Yes	Medium
MEM50-CPP	Do not access freed memory	No	High
MEM51-CPP	Properly deallocate dynamically allocated resources	Yes	High
MEM52-CPP	Detect and handle memory allocation errors	Yes	High
MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	No	High
MEM55-CPP	Honor replacement dynamic storage management requirements	No	
MEM57-CPP	Avoid using default operator new for over-aligned types	Yes	Medium
MSC50-CPP	Do not use the rand() function for generating pseudorandom numbers	Yes	Medium
MSC51-CPP	Ensure your random number generator is properly seeded	Yes	Medium
MSC52-CPP	Value-returning functions must return a value from all exit paths	Yes	Medium
MSC53-CPP	Do not return from a function declared [[noreturn]]	Yes	Medium
MSC54-CPP	A signal handler must be a plain old function	Yes	High
OOP50-CPP	Do not invoke virtual functions from constructors or destructors	Yes	Low
OOP51-CPP	Do not slice derived objects	Yes	Low
OOP52-CPP	Do not delete a polymorphic object without a virtual destructor	Yes	Low
OOP53-CPP	Write constructor member initializers in the canonical order	Yes	Medium
OOP54-CPP	Gracefully handle self-copy assignment	Yes	Low
OOP55-CPP	Do not use pointer-to-member operators to access nonexistent members	No	High
OOP56-CPP	Honor replacement handler requirements	Yes	Low

OOP57-CPP	Prefer special member functions and overloaded operators to C Standard Library functions	Yes	High
OOP58-CPP	Copy operations must not mutate the source object	Yes	Low
STR50-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator	Yes	High
STR51-CPP	Do not attempt to create a std::string from a null pointer	Yes	High
STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string	Yes	High
STR53-CPP	Range check element access	Yes	High