GRAMMATECH

SEI CERT-C++ RULES AND RECOMMENDATIONS

MAPPED TO CODESONAR® 6.2 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 6.2 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'smost security conscious organizations to detect, measure, analyze and resolve vulnerabilitiesfor 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 V6.2P0)

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

CERT-C++ Rules and Recommendations		Closely Mapped CodeSonar Warning Classes
CTR50-CPP	Guarantee that container indices and iterators are within thevalid range	Buffer Overrun Buffer Underrun Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun
CTR52-CPP	Guarantee that library functions do not overflow	Buffer Overrun Use of OemToAnsi Use of OemToChar Use of StrCatChainW Use of getopt Use of getpass Use of gets Use of getwd Use of recvmsg Use of strcat Use of strchr Use of strcmp Use of strcoll Use of strcpy Use of strcspn Use of strlen Use of strpbrk Use of strrchr Use of strspn Use of strstr Use of strtok Use of strtrns Use of syslog
DCL50-CPP	Do not define a C-style variadic function	Ellipsis
DCL51-CPP	Do not declare or define a reserved identifier	Declaration of Reserved Name Macro Name is C Keyword
DCL56-CPP	Avoid cycles during initialization of static objects	Initialization Cycle Unordered Initialization
DCL59-CPP	Do not define an unnamed namespace in a header file	Anonymous Namespace in Header File
DCL60-CPP	Obey the one-definition rule	Function Defined in Header File Object Defined in Header File
ERR50-CPP	Do not abruptly terminate the program	Use of Abort Use of exit

ERR52-CPP	Do not use setjmp() or longjmp()	Use of<setjmp.h> Use of longjmp Use of setjmp
ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived	Unreachable Catch
ERR57-CPP	Do not leak resources when handling exceptions	Leak
EXP53-CPP	Do not read uninitialized memory	Return Pointer to LocalUninitialized Variable
EXP54-CPP	Do not access an object outside of its lifetime	Use AfterClose Use After Free
EXP57-CPP	Do not cast or delete pointers to incomplete classes	Conversion: Pointer to Incomplete
FIO51-CPP	Close files when they are no longer needed	Leak
INT50-CPP	Do not cast to an out-of-range enumeration value	Cast Alters Value Coercion Alters Value
MEM50-CPP	Do not access freed memory	Use After Free
MEM51-CPP	Properly deallocate dynamically allocated resources	Double Free
MSC52-CPP	Value-returning functions must return a value from all exit paths	Missing Return Statement
MSC53-CPP	Do not return from a function declared [[noreturn]]	Return from noreturn
OOP50-CPP	Do not invoke virtual functions from constructors or destructors	Virtual Call in Constructor Virtual Call in Destructor
OOP51-CPP	Do not slice derived objects	Object Slicing
OOP52-CPP	Do not delete a polymorphic object without a virtual destructor	delete with Non-Virtual Destructor
OOP53-CPP	Write constructor member initializers in the canonical order	Out of Order Member Initializers
STR50-CPP	Guarantee that storage for strings has sufficient space for char-acter data and the null terminator	Buffer Overrun No Space For Null TerminatorType Overrun

SEI CERT C++ CODING STANDARD BROAD MAPPING (CODESONAR V6.2P0)

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

Warning classes that are also in the close mapping are shown in bold text.

CERT-C++ Rules and Recommendations		Broadly Mapped CodeSonar Warning Classes
CTR50-CPP	Guarantee that container indices and iterators are within the validrange	**Buffer Overrun** **Buffer Underrun** **Pointer Before Beginning of Object** **Pointer Past End of Object** **Tainted Buffer Access** **Type Overrun** **Type Underrun**
CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container	Use After Free
CTR52-CPP	Guarantee that library functions do not overflow	**Buffer Overrun** **Use of OemToAnsi** **Use of OemToChar** **Use of StrCatChainW** **Use of getopt** **Use of getpass** **Use of gets Use of getwd Use of recvmsgUse of strcat Use of strchr Use of strcmp Use of strcoll Use of strcpy Use of strcspn Use of strlen Use of strpbrk Use of strrchr Use of strspn Use of strstr Use of strtok Use of strtrns** **Use of syslog**
CTR53-CPP	Use valid iterator ranges	Buffer Overrun
CTR54-CPP	Do not subtract iterators that do not refer to the same container	Comparison of Unrelated Pointers Subtraction of Unrelated Pointers
DCL50-CPP	Do not define a C-style variadic function	**Ellipsis**

DCL51-CPP	Do not declare or define a reserved identifier	**Declaration of Reserved Name** **Macro Name is C Keyword**
DCL53-CPP	Use valid iterator ranges	Buffer Overrun
DCL54-CPP	Do not subtract iterators that do not refer to the same container	Comparison of Unrelated Pointers Subtraction of Unrelated Pointers
DCL56-CPP	Avoid cycles during initialization of static objects	**Initialization Cycle** **Unordered Initialization**
DCL59-CPP	Do not define an unnamed namespace in a header file	**Anonymous Namespace in Header File**
DCL60-CPP	Obey the one-definition rule	**Function Defined in Header File** **Object Defined in Header File**
ERR50-CPP	Do not abruptly terminate the program	**Use of abort** **Use of exit**
ERR52-CPP	Do not use setjmp() or longjmp()	**Use of <setjmp.h>** **Use of longjmp Use of setjmp**
ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived	**Unreachable Catch**
ERR56-CPP	Guarantee exception safety	Leak
ERR57-CPP	Do not leak resources when handling exceptions	**Leak**
ERR62-CPP	Detect errors when converting a string to a number	Use of atof Use of atoi Use of atol Use of atoll
EXP53-CPP	Do not read uninitialized memory	**Return Pointer to Local** **Uninitialized Variable**
EXP54-CPP	Do not access an object outside of its lifetime	**Use After Close** **Use After Free**
EXP57-CPP	Do not cast or delete pointers to incomplete classes	**Conversion: Pointer to Incomplete**
EXP62-CPP	Do not access the bits of an object representation that are not part of the object's value representation	Use of memcmp Use of memset
EXP63-CPP	Do not rely on the value of a moved-from object	Null Pointer Dereference
FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call	Input After Output Without Positioning Output After Input Without Positioning
FIO51-CPP	Close files when they are no longer needed	**Leak**
INT50-CPP	Do not cast to an out-of-range enumeration value	**Cast Alters Value** **Coercion Alters Value**
MEM50-CPP	Do not access freed memory	**Use After Free**

MEM51-CPP	Properly deallocate dynamically allocated resources	**Double Free** Free Non-Heap VariableLeak Type Mismatch
MSC52-CPP	Value-returning functions must return a value from all exit paths	**Missing Return Statement**
MSC53-CPP	Do not return from a function declared [[noreturn]]	**Return from noreturn**
OOP50-CPP	Do not invoke virtual functions from constructors or destructors	**Virtual Call in Constructor** **Virtual Call in Destructor**
OOP51-CPP	Do not slice derived objects	**Object Slicing**
OOP52-CPP	Do not delete a polymorphic object without a virtual destructor	**delete with Non-Virtual Destructor**
OOP53-CPP	Write constructor member initializers in the canonical order	**Out of Order Member Initializers**
OOP57-CPP	Prefer special member functions and overloaded operators to C Standard Library functions	Use of memcmp Use of memset
STR50-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator	**Buffer Overrun** **No Space For Null Terminator** **Type Overrun**
STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string	Use After Free
STR53-CPP	Range check element access	Buffer Overrun Buffer Underrun Tainted Buffer Access Type Overrun Type Underrun