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 secure coding 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 securityof software systems developed in the C programming language.

CodeSonar 6.2 includes a large number of warning classes that support checking for the CERT-Cguidelines. Every CodeSonar warning report includes the numbers of any CERT-C rules and recommendations that are closely mapped to the warning's class. (The close mapping for a warningclass 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-Crules 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://www.securecoding.cert.org/confluence/display/c/

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 thatare 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.

CodeSonar and CodeSentry are registered trademarks of GrammaTech, Inc.
© GrammaTech, Inc. All rights reserved.

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

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

CERT-C Rules and Recommendations		Closely Mapped CodeSonar Warning Classes
API00-C	Functions should validate their parameters	Unchecked Parameter Dereference
API07-C	Enforce type safety	Cast Alters Value Coercion Alters Value GlobalHandle on GMEM_FIXED Memory GlobalLock on GMEM_FIXED Memory GlobalUnlock on GMEM_FIXED Memory LocalHandle on LMEM_FIXED Memory LocalLock on LMEM_FIXED Memory LocalUnlock on LMEM_FIXED Memory Pool Mismatch Type Mismatch
ARR02-C	Explicitly specify array bounds, even if implicitly defined by an initializer	Declaration of Flexible Array Member
ARR30-C	Do not form or use out-of-bounds pointers or array sub-scripts	Buffer Overrun Buffer Underrun Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun
ARR36	Do not subtract or compare two pointers that do not refer to the same array	Comparison of Unrelated Pointers Subtraction of Unrelated Pointers
CON01-C	Acquire and release synchronization primitives in the samemodule, at the same level of abstraction	Missing Lock Acquisition Missing Lock Release
CON05-C	Do not perform operations that can block while holding alock	Blocking in Critical Section
CON07-C	Ensure that compound operations on shared variables are atomic	Data Race
CON32-C	Prevent data races when accessing bit-fields from multiple threads	Data Race
CON33-C	Avoid race conditions when using library functions	Use of rand Use of tmpnam Use of ttyname
CON34-C	Declare objects shared between threads with appropriate storage durations	Local Variable Passed to Thread
CON35-C	Avoid deadlock by locking in a predefined order	Conflicting Lock Order
CON37-C	Do not call signal() in a multithreaded program	Use of signal
CON40-C	Do not refer to an atomic variable twice in an expression	Multiple Accesses of Atomic

CON43-C	Do not allow data races in multithreaded code	Data Race
DCL00-C	Const-qualify immutable objects	Cast Removes const Qualifier Variable Could Be const
DCL01-C	Do not reuse variable names in subscopes	Non-distinct Identifiers: Nested Scope
DCL02-C	Use visually distinct identifiers	Typographically Ambiguous Identifiers
DCL04-C	Do not declare more than one variable per declaration	Multiple Declarations On Line
DCL05-C	Use typedefs of non-pointer types only	Pointer Type Inside Typedef

DCL07-C	Include the appropriate type information in function declar-ators	Incomplete Function Prototype
DCL11-C	Understand the type issues associated with variadic func-tions	Ellipsis
DCL13-C	Declare function parameters that are pointers to values notchanged by the function as const	Pointed-to Type Could Be const
DCL15-C	Declare file-scope objects or functions that do not need external linkage as static	Scope Could Be File Static
DCL16-C	Use “L,” not “l,” to indicate a long value	Confusing Literal Suffix
DCL18-C	Do not begin integer constants with 0 when specifying adecimal value	Octal Constant
DCL19-C	Minimize the scope of variables and functions	Scope Could Be File Static Scope Could Be Local Static
DCL20-C	Explicitly specify void when a function accepts no argu-ments	Incomplete Function Prototype
DCL23-C	Guarantee that mutually visible identifiers are unique	Global Variable Declared with Different Types Library Function Override Non-distinct Identifiers: External Names Non-distinct Identifiers: Macro/Macro Non-distinct Identifiers: Macro/Other Non-distinct Identifiers: Nested Scope Non-distinct Identifiers: Same Scope Non-unique Identifiers: External Name Non-unique Identifiers: Internal Name Non-unique Identifiers:Tag Non-unique Identifiers: Typedef
DCL30-C	Declare objects with appropriate storage durations	Return Pointer to Local
DCL37-C	Do not declare or define a reserved identifier	Declaration of Reserved Name
DCL40-C	Do not create incompatible declarations of the same func-tion or object	Inconsistent Function Declarations Inconsistent Object Declarations
ENV33-C	Do not call system()	Use of system
ERR33-C	Detect and handle standard library errors	Ignored Return Value
ERR34-C	Detect errors when converting a string to a number	Use of atof Use of atoi Use of atol Use of atoll
EXP00-C	Use parentheses for precedence of operation	Missing Parentheses
EXP05-C	Do not cast away a const qualification	Cast Removes const Qualifier
EXP08-C	Ensure pointer arithmetic is used correctly	Pointer Arithmetic Comparison of Unrelated Pointers Subtraction of Unrelated Pointers
EXP12-C	Do not ignore values returned by functions	Ignored Return Value
EXP14-C	Beware of integer promotion when performing bitwise oper-ations on integer types smaller than int	Risky Integer Promotion

EXP15-C	Do not place a semicolon on the same line as an if, for, orwhile statement	Empty Branch Statement Empty for Statement Empty if Statement Empty switch Statement Empty while Statement
EXP33-C	Do not read uninitialized memory	Uninitialized Variable
EXP34-C	Do not dereference null pointers	Null Pointer Dereference Unchecked Parameter Dereference
EXP37-C	Call functions with the correct number and type of arguments	Array Parameter Mismatch
EXP43-C	Avoid undefined behavior when using restrict-qualified pointers	Restrict Qualifier Used
EXP44-C	Do not rely on side effects in operands to sizeof, _Alignof, or _Generic	Side Effects in sizeof
EXP45-C	Do not perform assignments in selection statements	Assignment Result in ExpressionAssignment in Conditional Condition Contains Side Effects
EXP46-C	Do not use a bitwise operator with a Boolean-like operand	Inappropriate Operand Type
EXP47-C	Do not call va_arg with an argument of the incorrect type	Use of <stdarg.h> Feature
FIO01-C	Be careful using functions that use file names for identification	File System Race Condition Tainted Filename Use of GetTempFileName Use of mkstemp Use of mktemp Use of tmpfile Use of tmpnam
FIO02-C	Canonicalize path names originating from tainted sources	Tainted Filename
FIO06-C	Create files with appropriate access permissions	Use of CreateFile
FIO21-C	Do not create temporary files in shared directories	Use of GetTempFileName Use of mktemp Use of tmpfile Use of tmpnam
FIO30-C	Exclude user input from format strings	Format String Injection
FIO37-C	Do not assume that fgets() or fgetws() returns a nonempty string when successful	Buffer Underrun
FIO40-C	Reset strings on fgets() or fgetws() failure	Uninitialized Variable
FIO42-C	Close files when they are no longer needed	Leak
FIO45-C	Avoid TOCTOU race conditions while accessing files	File System Race Condition
FIO46-C	Do not access a closed file	Use After Close
FIO47-C	Use valid format strings	Format String InjectionFormat String Format String Type Error
FLP06-C	Convert integers to floating point for floating-point operations	Mismatched Operand Types

FLP30-C	Do not use floating-point variables as loop counters	Float-typed Loop Counter

FLP32-C	Prevent or detect domain and range errors in math functions	Arctangent Domain Error Argument Too High Argument Too Low Floating Point Domain Error Floating Point Range Error Gamma on Zero Logarithm on Negative Value Logarithm on Zero Raises FE_INVALID Undefined Power of Zero cosh on High Number cosh on Low Number sqrt on Negative Value
INT04-C	Enforce limits on integer values originating from taintedsources	Tainted Allocation Size Tainted Buffer Access Tainted Network Address Untrusted Network Host Untrusted Network Port
INT05-C	Do not use input functions to convert character data if theycannot handle all possible inputs	Negative Character Value
INT07-C	Use only explicitly signed or unsigned char type for numericvalues	Inappropriate Operand Type
INT09-C	Ensure enumeration constants map to unique values	Inconsistent Enumerator Initialization
INT12-C	Do not make assumptions about the type of a plain int bit-field when used in an expression	Bit-field Signedness Not Explicit
INT13-C	Use bitwise operators only on unsigned operands	Inappropriate Operand Type
INT18-C	Evaluate integer expressions in a larger size before comparing or assigning to that size	Cast Alters Value Coercion Alters Value Expression Value Widened by Assignment Expression Value Widened by Other Operand
INT31-C	Ensure that integer conversions do not result in lost ormisinterpreted data	Cast Alters Value Cast: Arithmetic Type/Void Pointer Coercion Alters Value Coercion: Integer Constant to Pointer Conversion: Pointer/Integer Tainted Buffer Access Truncation of Allocation SizeTruncation of Size
INT33-C	Ensure that division and remainder operations do not resultin divide-by-zero errors	Division By Zero Float Division By Zero
INT34-C	Do not shift an expression by a negative number of bits orby greater than or equal to the number of bits that exist inthe operand	Negative Shift Amount Shift Amount Exceeds Bit Width
INT36-C	Converting a pointer to integer or integer to pointer	Coercion: Integer Constant to PointerConversion: Pointer/Integer
MEM00-C	Allocate and free memory in the same module, at the samelevel of abstraction	Double Free

MEM07-C	Ensure that the arguments to calloc(), when multiplied, donot wrap	Multiplication Overflow of Allocation Size
MEM11-C	Do not assume infinite heap space	Leak Unreasonable Size Argument
MEM30-C	Do not access freed memory	Use After Free
MEM31-C	Free dynamically allocated memory when no longer needed	Leak
MEM35-C	Allocate sufficient memory for an object	Addition Overflow of Allocation Size Integer Overflow of Allocation Size Multiplication Overflow of Allocation Size Subtraction Underflow of Allocation Size
MSC00-C	Compile cleanly at high warning levels	Not All Warnings Are Enabled Warnings Not Treated As Errors
MSC06-C	Beware of compiler optimizations	Use of memset
MSC07-C	Detect and remove dead code	Redundant Condition Unexercised Call Unexercised Computation Unexercised Conditional Unexercised Control Flow Unexercised Data Flow Unreachable Call Unreachable Computation Unreachable Conditional Unreachable Control Flow Unreachable Data Flow
MSC11-C	Incorporate diagnostic tests using assertions	Not Enough Assertions

MSC12-C	Detect and remove code that has no effect or is never executed	Empty Branch Statement Empty for Statement Empty if Statement Empty switch Statement Empty while Statement Function Call Has No Effect Redundant Condition Unexercised Call Unexercised Computation Unexercised Conditional Unexercised Control Flow Unexercised Data Flow Unreachable Call Unreachable Computation Unreachable Conditional Unreachable Control Flow Unreachable Data Flow Unused Label Unused Macro Unused Parameter Unused Tag Unused Type Unused Variable Useless Assignment
MSC13-C	Detect and remove unused values	Unused Value
MSC17-C	Finish every set of statements associated with a case labelwith a break statement	Missing break
MSC18-C	Be careful while handling sensitive data, such as pass-words, in program code	Plaintext Transmission of Password
MSC20-C	Do not use a switch statement to transfer control into acomplex block	Misplaced case
MSC21-C	Use robust loop termination conditions	Potential Unbounded Loop
MSC22-C	Use the setjmp(), longjmp() facility securely	Use of longjmp Use of setjmp
MSC24-C	Do not use deprecated or obsolescent functions	Use of Load Module Use of MoveFile Use of WinExec Use of cuserid Use of drem Use of gamma
MSC30-C	Do not use the rand() function for generating pseudoran-dom numbers	Use of rand
MSC37-C	Ensure that control never reaches the end of a non-voidfunction	Missing Return Statement
MSC41-C	Never hard code sensitive information	Hardcoded Authentication Hardcoded Crypto Key Hardcoded Crypto Salt Hardcoded DNS Name

POS05-C	Limit access to files by creating a jail	Use of chroot chroot without chdir
POS33-C	Do not use vfork()	Use of vfork
POS49-C	When data must be accessed by multiple threads, providea mutex and guarantee no adjacent data is also accessed	Data Race
POS51-C	Avoid deadlock with POSIX threads by locking in predefined order	Conflicting Lock Order
POS52-C	Do not perform operations that can block while holding aPOSIX lock	Blocking in Critical Section
PRE00-C	Prefer inline or static functions to function-like macros	Function-Like Macro
PRE02-C	Macro replacement lists should be parenthesized	Macro Does Not End With } or) Macro Does Not Start With { or (
PRE05-C	Understand macro replacement when concatenating to-kens or performing stringification	Macro Uses # Operator
PRE11-C	Do not conclude macro definitions with a semicolon	Macro Does Not End With } or)
PRE30-C	Do not create a universal character name through concat-enation	## Follows # Operator Macro Uses ## Operator
PRE32-C	Do not use preprocessor directives in invocations of func-tion-like macros	Preprocessing Directives in Macro Argument
SIG02-C	Avoid using signals to implement normal functionality	Use of signal
SIG31-C	Do not access shared objects in signal handlers	Data Race
SIG34-C	Do not call signal() from within interruptible signal handlers	Use of signal
STR00-C	Represent characters using an appropriate type	Negative Character Value
STR02-C	Sanitize data passed to complex subsystems	Command Injection Format String Injection LDAP Injection Library Injection SQL Injection Untrusted Library Load Untrusted Process Creation
STR03-C	Do not inadvertently truncate a string	No Space For Null Terminator
STR05-C	Use pointers to const when referring to string literals	Non-const String Literal

STR07-C	Use the bounds-checking interfaces for string manipulation	Use of OemToAnsi Use of OemToChar Use of StrCatChainW 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

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

The following table contains CodeSonar warning classes that are broadly mapped to CERT-C rulesand 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
API00-C	Functions should validate their parameters	Buffer Overrun Integer Overflow of Allocation Size Null Pointer Dereference Tainted Buffer Access **Unchecked Parameter Dereference**
API07-C	Enforce type safety	**Cast Alters Value** **Coercion Alters Value** **GlobalHandle on GMEM_FIXED Memory** **GlobalLock on GMEM_FIXED Memory** **GlobalUnlock on GMEM_FIXED Memory** **LocalHandle on LMEM_FIXED Memory** **LocalLock on LMEM_FIXED Memory** **LocalUnlock on LMEM_FIXED Memory** **Pool Mismatch** **Type Mismatch**
ARR02-C	Explicitly specify array bounds, even if implicitly defined by an initializer	**Declaration of Flexible Array Member**
ARR30-C	Do not form or use out-of-bounds pointers or arraysubscripts	**Buffer Overrun** **Buffer Underrun** **Pointer Before Beginning of Object** **Pointer Past End of Object** **Tainted Buffer Access** **Type Overrun** **Type Underrun**
ARR36-C	Do not subtract or compare two pointers that do not refer to the same array	**Comparison of Unrelated Pointers** **Subtraction of Unrelated Pointers**
ARR38-C	Guarantee that library functions do not form invalidpointers	Buffer Overrun Buffer Underrun
CON01-C	Acquire and release synchronization primitives in thesame module, at the same level of abstraction	**Missing Lock Acquisition** **Missing Lock Release**
CON05-C	Do not perform operations that can block whileholding a lock	**Blocking in Critical Section**
CON06-C	Ensure that every mutex outlives the data it protects	Data Race Use After Free
CON07-C	Ensure that compound operations on shared variables are atomic	**Data Race**
CON32-C	Prevent data races when accessing bit-fields from multiple threads	**Data Race** Multiple Accesses of Atomic

CON33-C	Avoid race conditions when using library functions	**Use of rand Use of tmpnamUse of ttyname**
CON34-C	Declare objects shared between threads with appropriate storage durations	**Local Variable Passed to Thread**
CON35-C	Avoid deadlock by locking in a predefined order	**Conflicting Lock Order**
CON36-C	Wrap functions that can spuriously wake up in a loop	Use of cnd_wait

CON37-C	Do not call signal() in a multithreaded program	**Use of signal**
CON40-C	Do not refer to an atomic variable twice in an expression	**Multiple Accesses of Atomic**
CON43-C	Do not allow data races in multithreaded code	**Data Race** Multiple Accesses of Atomic
DCL00-C	Const-qualify immutable objects	**Cast Removes const Qualifier**
DCL01-C	Do not reuse variable names in subscopes	**Non-distinct Identifiers: Nested Scope**
DCL02-C	Use visually distinct identifiers	**Typographically Ambiguous Identifiers**
DCL04-C	Do not declare more than one variable per declara-tion	**Multiple Declarations On Line**
DCL05-C	Use typedefs of non-pointer types only	**Pointer Type Inside Typedef**
DCL07-C	Include the appropriate type information in functiondeclarators	**Incomplete Function Prototype**
DCL11-C	Understand the type issues associated with variadicfunctions	**Ellipsis**
DCL13-C	Declare function parameters that are pointers tovalues not changed by the function as const	**Pointed-to Type Could Be const**
DCL15-C	Declare file-scope objects or functions that do not need external linkage as static	**Scope Could Be File Static**
DCL16-C	Use “L,” not “l,” to indicate a long value	**Confusing Literal Suffix**
DCL18-C	Do not begin integer constants with 0 when specify-ing a decimal value	**Octal Constant**
DCL19-C	Minimize the scope of variables and functions	**Scope Could Be File Static** **Scope Could Be Local Static**
DCL20-C	Explicitly specify void when a function accepts noarguments	**Incomplete Function Prototype**
DCL23-C	Guarantee that mutually visible identifiers are unique	**Global Variable Declared with Different Types** **Library Function Override** **Non-distinct Identifiers: External Names** **Non-distinct Identifiers: Macro/Macro** **Non-distinct Identifiers: Macro/Other** **Non-distinct Identifiers: Nested Scope** **Non-distinct Identifiers: Same Scope** **Non-unique Identifiers: External Name** **Non-unique Identifiers: Internal Name** **Non-unique Identifiers: Tag** **Non-unique Identifiers: Typedef**
DCL30-C	Declare objects with appropriate storage durations	**Return Pointer to Local**
DCL37-C	Do not declare or define a reserved identifier	**Declaration of Reserved Name**
DCL40-C	Do not create incompatible declarations of the samefunction or object	Implicit Function Declaration **Inconsistent Function Declarations** **Inconsistent Object Declarations**
ENV01-C	Do not make assumptions about the size of anenvironment variable	Buffer Overrun Type Overrun
ENV33-C	Do not call system()	**Use of system**

ENV34-C	Do not store pointers returned by certain functions	Use After Free
ERR30-C	Set errno to zero before calling a library function known to set errno, and check errno only after the function returns a value indicating failure	Ignored Return Value Redundant Condition
ERR33-C	Detect and handle standard library errors	**Ignored Return Value**

ERR34-C	Detect errors when converting a string to a number	**Use of atof** **Use of atoi** **Use of atol** **Use of atoll**
EXP00-C	Use parentheses for precedence of operation	**Missing Parentheses**
EXP05-C	Do not cast away a const qualification	**Cast Removes const Qualifier**
EXP08-C	Ensure pointer arithmetic is used correctly	Buffer Overrun Buffer Underrun **Pointer Arithmetic** Pointer Before Beginning of ObjectPointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun **Comparison of Unrelated Pointers** **Subtraction of Unrelated Pointers**
EXP12-C	Do not ignore values returned by functions	**Ignored Return Value**
EXP14-C	Beware of integer promotion when performing bit-wise operations on integer types smaller than int	**Risky Integer Promotion**
EXP15-C	Do not place a semicolon on the same line as an if,for, or while statement	**Empty Branch Statement** **Empty for Statement** **Empty if Statement** **Empty switch Statement** **Empty while Statement**
EXP33-C	Do not read uninitialized memory	**Uninitialized Variable**
EXP34-C	Do not dereference null pointers	**Null Pointer Dereference** Null Test After Dereference **Unchecked Parameter Dereference**
EXP37-C	Call functions with the correct number and type ofarguments	**Array Parameter Mismatch**
EXP42-C	Do not compare padding data	Use of memcmp
EXP43-C	Avoid undefined behavior when using restrict-quali-fied pointers	**Restrict Qualifier Used**
EXP44-C	Do not rely on side effects in operands to sizeof, _Alignof, or _Generic	**Side Effects in sizeof**
EXP45-C	Do not perform assignments in selection statements	**Assignment Result in ExpressionAssignment in Conditional Condition Contains Side Effects**
EXP46-C	Do not use a bitwise operator with a Boolean-likeoperand	**Inappropriate Operand Type**
EXP47-C	Do not call va_arg with an argument of the incorrecttype	**Use of <stdarg.h> Feature**

FIO01-C	Be careful using functions that use file names foridentification	**File System Race Condition** **Tainted Filename** **Use of GetTempFileName** **Use of mkstemp** **Use of mktemp** **Use of tmpfile** **Use of tmpnam**
FIO02-C	Canonicalize path names originating from taintedsources	**Tainted Filename**
FIO06-C	Create files with appropriate access permissions	**Use of CreateFile**

FIO21-C	Do not create temporary files in shared directories	**Use of GetTempFileName** **Use of mktemp** **Use of tmpfile** **Use of tmpnam**
FIO24-C	Do not open a file that is already open	File System Race Condition
FIO30-C	Exclude user input from format strings	Format String **Format String Injection**
FIO34-C	Distinguish between characters read from a file and EOF or WEOF	Coercion Alters Value
FIO37-C	Do not assume that fgets() or fgetws() returns a nonempty string when successful	**Buffer Underrun**
FIO39-C	Do not alternately input and output from a stream without an intervening flush or positioning call	Input After Output Without Positioning Output After Input Without Positioning
FIO40-C	Reset strings on fgets() or fgetws() failure	**Uninitialized Variable**
FIO42-C	Close files when they are no longer needed	**Leak**
FIO45-C	Avoid TOCTOU race conditions while accessing files	**File System Race Condition**
FIO46-C	Do not access a closed file	**Use After Close**
FIO47-C	Use valid format strings	Format String **Format String Injection**
FLP06-C	Convert integers to floating point for floating-point operations	**Mismatched Operand Types**
FLP30-C	Do not use floating-point variables as loop counters	**Float-typed Loop Counter**
FLP32-C	Prevent or detect domain and range errors in mathfunctions	**Arctangent Domain Error** **Argument Too High** **Argument Too Low** **Floating Point Domain Error** **Floating Point Range Error** **Gamma on Zero** **Logarithm on Negative Value** **Logarithm on Zero** **Raises FE_INVALID** **Undefined Power of Zero** **cosh on High Number** **cosh on Low Number** **sqrt on Negative Value**
INT02-C	Understand integer conversion rules	Cast Alters Value Coercion Alters Value Truncation of Allocation Size Truncation of Size
INT04-C	Enforce limits on integer values originating fromtainted sources	**Tainted Allocation Size** **Tainted Buffer Access** **Tainted Network Address** **Untrusted Network Host** **Untrusted Network Port**

INT05-C	Do not use input functions to convert character dataif they cannot handle all possible inputs	**Negative Character Value**
INT07-C	Use only explicitly signed or unsigned char type fornumeric values	**Inappropriate Operand Type**
INT08-C	Verify that all integer values are in range	**Overflow of Size** **Multiplication Overflow of Size** **Subtraction Underflow of Size**
INT09-C	Ensure enumeration constants map to uniquevalues	**Inconsistent Enumerator Initialization**

INT12-C	Do not make assumptions about the type of a plain int bit-field when used in an expression	**Bit-field Signedness Not Explicit**
INT13-C	Use bitwise operators only on unsigned operands	**Inappropriate Operand Type**
INT18-C	Evaluate integer expressions in a larger size beforecomparing or assigning to that size	Addition Overflow of Allocation SizeAddition Overflow of Size **Cast Alters Value** **Coercion Alters Value** **Expression Value Widened by Assignment** **Expression Value Widened by Other Operand**Integer Overflow of Allocation Size Multiplication Overflow of Allocation SizeMultiplication Overflow of Size Subtraction Underflow of Allocation SizeSubtraction Underflow of Size Truncation of Allocation Size Truncation of Size
INT30-C	Ensure that unsigned integer operations do notwrap	Addition Overflow of Allocation SizeAddition Overflow of Size Integer Overflow of Allocation Size Multiplication Overflow of Allocation SizeMultiplication Overflow of Size Subtraction Underflow of Allocation SizeSubtraction Underflow of Size Unreasonable Size Argument
INT31-C	Ensure that integer conversions do not result in lostor misinterpreted data	**Cast Alters Value** **Cast: Arithmetic Type/Void Pointer** **Coercion Alters Value** **Coercion: Integer Constant to Pointer** **Conversion: Pointer/Integer** **Tainted Buffer Access** **Truncation of Allocation Size** **Truncation of Size**
INT32-C	Ensure that operations on signed integers do not result in overflow	Addition Overflow of Allocation SizeAddition Overflow of Size Integer Overflow of Allocation Size Multiplication Overflow of Allocation SizeMultiplication Overflow of Size Subtraction Underflow of Allocation SizeSubtraction Underflow of Size Unreasonable Size Argument
INT33-C	Ensure that division and remainder operations donot result in divide-by-zero errors	**Division By Zero** **Float Division By Zero**
INT34-C	Do not shift an expression by a negative number of bits or by greater than or equal to the number ofbits that exist in the operand	**Negative Shift Amount** **Shift Amount Exceeds Bit Width**
INT36-C	Converting a pointer to integer or integer to pointer	**Coercion: Integer Constant to Pointer** **Conversion: Pointer/Integer**
MEM00-C	Allocate and free memory in the same module, atthe same level of abstraction	**Double Free** Leak

MEM01-C	Store a new value in pointers immediately after free()	Double Free Use After Free

MEM05-C	Avoid large stack allocations	Tainted Allocation Size Unreasonable Size Argument
MEM07-C	Ensure that the arguments to calloc(), when multi-plied, do not wrap	**Multiplication Overflow of Allocation Size**
MEM11-C	Do not assume infinite heap space	**Leak** Tainted Allocation Size **Unreasonable Size Argument**
MEM30-C	Do not access freed memory	**Use After Free**
MEM31-C	Free dynamically allocated memory when no longerneeded	**Leak**
MEM35-C	Allocate sufficient memory for an object	**Addition Overflow of Allocation Size** Buffer Overrun Buffer Underrun **Integer Overflow of Allocation Size** **Multiplication Overflow of Allocation Size** Pointer Before Beginning of Object Pointer Past End of Object **Subtraction Underflow of Allocation Size** Tainted Allocation Size Tainted Buffer Access Truncation of Allocation SizeType Overrun Type Underrun Unreasonable Size Argument
MSC00-C	Compile cleanly at high warning levels	**Not All Warnings Are Enabled** **Warnings Not Treated As Errors**
MSC06-C	Beware of compiler optimizations	**Use of memset**
MSC07-C	Detect and remove dead code	**Redundant Condition** **Unexercised Call** **Unexercised Computation** **Unexercised Conditional** **Unexercised Control Flow** **Unexercised Data Flow** **Unreachable Call** **Unreachable Computation** **Unreachable Conditional** **Unreachable Control Flow** **Unreachable Data Flow**
MSC11-C	Incorporate diagnostic tests using assertions	**Not Enough Assertions**

MSC12-C	Detect and remove code that has no effect or is never executed	**Empty Branch Statement** **Empty for Statement** **Empty if Statement** **Empty switch Statement** **Empty while Statement** **Function Call Has No Effect** **Redundant Condition** **Unexercised Call** **Unexercised Computation** **Unexercised Conditional** **Unexercised Control Flow** **Unexercised Data Flow** **Unreachable Call** **Unreachable Computation** **Unreachable Conditional** **Unreachable Control Flow** **Unreachable Data Flow** **Unused Label** **Unused Macro** **Unused Parameter** **Unused Tag** **Unused Type** **Unused Variable** **Useless Assignment**
MSC13-C	Detect and remove unused values	**Unused Value**
MSC17-C	Finish every set of statements associated with acase label with a break statement	**Missing break**
MSC18-C	Be careful while handling sensitive data, such aspasswords, in program code	**Hardcoded Authentication** **Hardcoded Crypto Key** **Hardcoded Crypto Salt** **Plaintext Storage of Password** **Plaintext Transmission of Password**
MSC20-C	Do not use a switch statement to transfer controlinto a complex block	**Misplaced case**
MSC21-C	Use robust loop termination conditions	High Risk Loop **Potential Unbounded Loop**
MSC22-C	Use the setjmp(), longjmp() facility securely	**Use of longjmp** **Use of setjmp**
MSC24-C	Do not use deprecated or obsolescent functions	**Use of LoadModule** **Use of MoveFile** **Use of WinExec** **Use of cuserid** **Use of drem** **Use of gamma**
MSC30-C	Do not use the rand() function for generating pseu-dorandom numbers	**Use of rand**
MSC37-C	Ensure that control never reaches the end of a non-void function	**Missing Return Statement**
MSC41-C	Never hard code sensitive information	**Hardcoded Authentication** **Hardcoded Crypto Key** **Hardcoded Crypto Salt** **Hardcoded DNS Name**

POS05-C	Limit access to files by creating a jail	**Use of chroot** **chroot without chdir**
POS33-C	Do not use vfork()	**Use of vfork**
POS38-C	Beware of race conditions when using fork and file descriptors	Use of fork
POS49-C	When data must be accessed by multiple threads, provide a mutex and guarantee no adjacent data isalso accessed	**Data Race** Multiple Accesses of Atomic
POS51-C	Avoid deadlock with POSIX threads by locking in predefined order	**Conflicting Lock Order**
POS52-C	Do not perform operations that can block whileholding a POSIX lock	**Blocking in Critical Section**
POS54-C	Detect and handle POSIX library errors	Ignored Return Value
PRE02-C	Macro replacement lists should be parenthesized	**Macro Does Not End With } or)** **Macro Does Not Start With { or (**
PRE05-C	Understand macro replacement when concatenat-ing tokens or performing stringification	**Macro Uses # Operator** Macro Uses ## Operator
PRE11-C	Do not conclude macro definitions with a semicolon	**Macro Does Not End With } or)**
PRE30-C	Do not create a universal character name throughconcatenation	**## Follows # Operator** **Macro Uses ## Operator**
PRE32-C	Do not use preprocessor directives in invocations offunction-like macros	**Preprocessing Directives in Macro Argument**
SIG00-C	Mask signals handled by noninterruptible signalhandlers	Use of signal
SIG01-C	Understand implementation-specific details regard-ing signal handler persistence	Use of signal
SIG02-C	Avoid using signals to implement normal function-ality	**Use of signal**
SIG31-C	Do not access shared objects in signal handlers	**Data Race**
SIG34-C	Do not call signal() from within interruptible signalhandlers	**Use of signal**
STR00-C	Represent characters using an appropriate type	**Negative Character Value**
STR02-C	Sanitize data passed to complex subsystems	**Command Injection** **Format String Injection** **LDAP Injection** **Library Injection** **SQL Injection** **Untrusted Library Load** **Untrusted Process Creation**
STR03-C	Do not inadvertently truncate a string	**No Space For Null Terminator**
STR04-C	Use plain char for characters in the basic characterset	Inappropriate Assignment Type Inappropriate Character ArithmeticInappropriate Operand Type Mismatched Operand Types
STR05-C	Use pointers to const when referring to string	**Non-const String Literal**

	literals	

STR07-C	Use the bounds-checking interfaces for stringmanipulation	**Use of OemToAnsi** **Use of OemToChar** **Use of StrCatChainW** **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**
STR31-C	Guarantee that storage for strings has sufficient space for character data and the null terminator	**Buffer Overrun** **No Space For Null Terminator** **Type Overrun**
STR32-C	Do not pass a non-null-terminated character se-quence to a library function that expects a string	**Unterminated C String**
STR34-C	Cast characters to unsigned char before convertingto larger integer sizes	**Negative Character Value**
STR37-C	Arguments to character-handling functions must berepresentable as an unsigned char	**Negative Character Value**
WIN00-C	Be specific when dynamically loading libraries	**Use of AfxLoadLibrary** **Use of CoLoadLibrary** **Use of LoadLibrary**
WIN02-C	Restrict privileges when spawning child processes	**Use of CreateProcess** **Use of CreateThread**
WIN30-C	Properly pair allocation and deallocation functions	Pool MismatchType Mismatch