Polyspace Bug Finder

This page was automatically generated and should not be edited.

The information on this page was provided by outside contributors and has not been verified by SEI CERT.

The table below can be re-ordered, by clicking column headers.

Tool Version: R2018a

Checker	Guideline
Abnormal termination of exit handler	ENV32-C. All exit handlers must return normally
Absorption of float operand	FLP00-C. Understand the limitations of floating-point numbers
Accessing object with temporary lifetime	EXP35-C. Do not modify objects with temporary lifetime
Alignment changed after memory allocation	MEM36-C. Do not modify the alignment of objects by calling realloc()
Alternating input and output from a stream without flush or positioning call	FIO39-C. Do not alternately input and output from a stream without an intervening flush or positioning call
Arithmetic operation with NULL pointer	EXP34-C. Do not dereference null pointers
Array access out of bounds	ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
Array access out of bounds	ARR38-C. Guarantee that library functions do not form invalid pointers
Array access out of bounds	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Array access out of bounds	API02-C. Functions that read or write to or from an array should take an argument to specify the source or target size
Array access out of bounds	ARR00-C. Understand how arrays work
Array access out of bounds	MSC15-C. Do not depend on undefined behavior
Array access with tainted index	ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
Array access with tainted index	API02-C. Functions that read or write to or from an array should take an argument to specify the source or target size
Array access with tainted index	INT04-C. Enforce limits on integer values originating from tainted sources
Bad file access mode or status	EXP37-C. Call functions with the correct number and type of arguments
Bad file access mode or status	FIO11-C. Take care when specifying the mode parameter of fopen()
Bad order of dropping privileges	POS36-C. Observe correct revocation order while relinquishing privileges
Bitwise and arithmetic operation on the same data	INT14-C. Avoid performing bitwise and arithmetic operations on the same data
Bitwise operation on negative value	INT13-C. Use bitwise operators only on unsigned operands
Buffer overflow from incorrect string format specifier	ARR38-C. Guarantee that library functions do not form invalid pointers
Buffer overflow from incorrect string format specifier	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Buffer overflow from incorrect string format specifier	STR03-C. Do not inadvertently truncate a string
Call to memset with unintended value	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
Character value absorbed into EOF	FIO34-C. Distinguish between characters read from a file and EOF or WEOF
Closing a previously closed resource	FIO46-C. Do not access a closed file
Command executed from an externally controlled path	ENV33-C. Do not call system()
Command executed from externally controlled path	STR02-C. Sanitize data passed to complex subsystems
Constant block cipher initialization vector	MSC18-C. Be careful while handling sensitive data, such as passwords, in program code
Constant cipher key	MSC18-C. Be careful while handling sensitive data, such as passwords, in program code
Copy of overlapping memory	EXP43-C. Avoid undefined behavior when using restrict-qualified pointers
Copy of overlapping memory	MSC15-C. Do not depend on undefined behavior
Data race	CON32-C. Prevent data races when accessing bit-fields from multiple threads

Data race	CON43-C. Do not allow data races in multithreaded code
Data race	POS49-C. When data must be accessed by multiple threads, provide a mutex and guarantee no adjacent data is also accessed
Data race	CON09-C. Avoid the ABA problem when using lock-free algorithms
Data race through standard library function call	CON33-C. Avoid race conditions when using library functions
Dead code	MSC01-C. Strive for logical completeness
Dead code	MSC07-C. Detect and remove dead code
Dead code	MSC12-C. Detect and remove code that has no effect or is never executed
Deadlock	CON35-C. Avoid deadlock by locking in a predefined order
Deadlock	POS51-C. Avoid deadlock with POSIX threads by locking in predefined order
Deallocation of previously deallocated pointer	MEM30-C. Do not access freed memory
Deallocation of previously deallocated pointer	MEM00-C. Allocate and free memory in the same module, at the same level of abstraction
Declaration mismatch	DCL40-C. Do not create incompatible declarations of the same function or object
Declaration mismatch	EXP37-C. Call functions with the correct number and type of arguments
Declaration mismatch	MSC15-C. Do not depend on undefined behavior
Destination buffer overflow in string manipulation	ARR38-C. Guarantee that library functions do not form invalid pointers
Destination buffer overflow in string manipulation	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Destination buffer overflow in string manipulation	STR38-C. Do not confuse narrow and wide character strings and functions
Destination buffer overflow in string manipulation	ENV01-C. Do not make assumptions about the size of an environment variable
Destination buffer overflow in string manipulation	STR07-C. Use the bounds-checking interfaces for string manipulation
Destination buffer underflow in string manipulation	ARR38-C. Guarantee that library functions do not form invalid pointers
Destruction of locked mutex	CON31-C. Do not destroy a mutex while it is locked
Destruction of locked mutex	POS48-C. Do not unlock or destroy another POSIX thread's mutex
Deterministic random output from constant seed	MSC32-C. Properly seed pseudorandom number generators
Double lock	CON01-C. Acquire and release synchronization primitives in the same module, at the same level of abstraction
Double unlock	CON01-C. Acquire and release synchronization primitives in the same module, at the same level of abstraction
Environment pointer invalidated by previous operation	ENV31-C. Do not rely on an environment pointer following an operation that may invalidate it
Errno not checked	ERR33-C. Detect and handle standard library errors
Errno not reset	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
Execution of externally controlled command	ENV33-C. Do not call system()
Execution of externally controlled command	STR02-C. Sanitize data passed to complex subsystems
File access between time of check and use (TOCTOU)	FIO45-C. Avoid TOCTOU race conditions while accessing files
File access between time of check and use (TOCTOU)	POS35-C. Avoid race conditions while checking for the existence of a symbolic link
File access between time of check and use (TOCTOU)	FIO01-C. Be careful using functions that use file names for identification
File descriptor exposure to child process	POS38-C. Beware of race conditions when using fork and file descriptors
File manipulation after chroot() without chdir("/")	POS05-C. Limit access to files by creating a jail
Float conversion overflow	FLP34-C. Ensure that floating-point conversions are within range of the new type
Float conversion overflow	FLP03-C. Detect and handle floating-point errors
Float division by zero	FLP03-C. Detect and handle floating-point errors
Float overflow	FLP03-C. Detect and handle floating-point errors
Float overflow	FLP06-C. Convert integers to floating point for floating-point operations
Floating point comparison with equality operators	FLP02-C. Avoid using floating-point numbers when precise computation is needed
Format string specifiers and arguments mismatch	EXP37-C. Call functions with the correct number and type of arguments
Format string specifiers and arguments mismatch	DCL10-C. Maintain the contract between the writer and caller of variadic functions
Format string specifiers and arguments mismatch	DCL11-C. Understand the type issues associated with variadic functions
Format string specifiers and arguments mismatch	INT00-C. Understand the data model used by your implementation(s)

Format string specifiers and arguments mismatch	MSC15-C. Do not depend on undefined behavior
Function called from signal-handler not asynchronous safe	SIG30-C. Call only asynchronous-safe functions within signal handlers
Function called from signal-handler not asynchronous safe (strict)	SIG30-C. Call only asynchronous-safe functions within signal handlers
Hard coded buffer size	DCL06-C. Use meaningful symbolic constants to represent literal values
Hard coded loop boundary	DCL06-C. Use meaningful symbolic constants to represent literal values
Hard-coded object size used to manipulate memory	EXP09-C. Use sizeof to determine the size of a type or variable
Improper array initialization	ARR00-C. Understand how arrays work
Improper array initialization	ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer
Incorrect data type passed to va_arg	EXP47-C. Do not call va_arg with an argument of the incorrect type
Incorrect pointer scaling	ARR39-C. Do not add or subtract a scaled integer to a pointer
Incorrect pointer scaling	EXP08-C. Ensure pointer arithmetic is used correctly
Information leak via structure padding	DCL39-C. Avoid information leakage when passing a structure across a trust boundary
Inline constraint not respected	MSC40-C. Do not violate constraints
Integer conversion overflow	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
Integer conversion overflow	FLP34-C. Ensure that floating-point conversions are within range of the new type
Integer conversion overflow	INT02-C. Understand integer conversion rules
Integer conversion overflow	INT12-C. Do not make assumptions about the type of a plain int bit-field when used in an expression
Integer conversion overflow	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
Integer division by zero	INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors
Integer overflow	INT32-C. Ensure that operations on signed integers do not result in overflow
Integer overflow	INT00-C. Understand the data model used by your implementation(s)
Integer overflow	INT02-C. Understand integer conversion rules
Integer overflow	INT08-C. Verify that all integer values are in range
Integer overflow	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
Integer overflow	MSC15-C. Do not depend on undefined behavior
Invalid assumptions about memory organization	ARR37-C. Do not add or subtract an integer to a pointer to a non-array object
Invalid file position	FIO44-C. Only use values for fsetpos() that are returned from fgetpos()
Invalid free of pointer	MEM34-C. Only free memory allocated dynamically
Invalid free of pointer	MEM00-C. Allocate and free memory in the same module, at the same level of abstraction
Invalid use of = operator	EXP45-C. Do not perform assignments in selection statements
Invalid use of standard library floating point routine	FLP32-C. Prevent or detect domain and range errors in math functions
Invalid use of standard library floating point routine	FLP03-C. Detect and handle floating-point errors
Invalid use of standard library integer routine	STR37-C. Arguments to character-handling functions must be representable as an unsigned char
Invalid use of standard library memory routine	EXP34-C. Do not dereference null pointers
Invalid use of standard library memory routine	ARR38-C. Guarantee that library functions do not form invalid pointers
Invalid use of standard library memory routine	ARR39-C. Do not add or subtract a scaled integer to a pointer
Invalid use of standard library memory routine	API00-C. Functions should validate their parameters
Invalid use of standard library memory routine	EXP08-C. Ensure pointer arithmetic is used correctly
Invalid use of standard library memory routine	MSC15-C. Do not depend on undefined behavior
Invalid use of standard library routine	API00-C. Functions should validate their parameters
Invalid use of standard library routine	MSC15-C. Do not depend on undefined behavior
Invalid use of standard library string routine	ARR38-C. Guarantee that library functions do not form invalid pointers
Invalid use of standard library string routine	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Invalid use of standard library string routine	STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string
Invalid use of standard library string routine	MEM30-C. Do not access freed memory
Invalid use of standard library string routine	API00-C. Functions should validate their parameters

Invalid use of standard library string routine	MSC15-C. Do not depend on undefined behavior
Invalid va_list argument	MSC39-C. Do not call va_arg() on a va_list that has an indeterminate value
Library loaded from externally controlled path	STR02-C. Sanitize data passed to complex subsystems
Library loaded from externally controlled path	WIN00-C. Be specific when dynamically loading libraries
Load of library from a relative path can be controlled by an external actor	WIN00-C. Be specific when dynamically loading libraries
Loop bounded with tainted value	INT04-C. Enforce limits on integer values originating from tainted sources
Loop bounded with tainted value	MSC21-C. Use robust loop termination conditions
Memory allocated with tainted size	ARR32-C. Ensure size arguments for variable length arrays are in a valid range
Memory allocation with tainted size	MEM35-C. Allocate sufficient memory for an object
Memory allocation with tainted size	INT04-C. Enforce limits on integer values originating from tainted sources
Memory allocation with tainted size	MEM07-C. Ensure that the arguments to calloc(), when multiplied, do not wrap
Memory allocation with tainted size	MEM10-C. Define and use a pointer validation function
Memory allocation with tainted size	MEM11-C. Do not assume infinite heap space
Memory comparison of float-point values	FLP37-C. Do not use object representations to compare floating-point values
Memory comparison of padding data	EXP42-C. Do not compare padding data
Memory leak	MEM31-C. Free dynamically allocated memory when no longer needed
Memory leak	MEM11-C. Do not assume infinite heap space
Memory leak	MEM12-C. Consider using a goto chain when leaving a function on error when using and releasing resources
Mismatch between data length and size	ARR38-C. Guarantee that library functions do not form invalid pointers
Mismatched alloc/dealloc functions on Windows	WIN30-C. Properly pair allocation and deallocation functions
MISRA C:2012 Dir 4.5	DCL02-C. Use visually distinct identifiers
MISRA C:2012 Dir 4.6	INT00-C. Understand the data model used by your implementation(s)
MISRA C:2012 Dir 4.8	DCL12-C. Implement abstract data types using opaque types
MISRA C:2012 Dir 4.9	PRE00-C. Prefer inline or static functions to function-like macros
MISRA C:2012 Dir 4.10	PRE06-C. Enclose header files in an include guard
MISRA C:2012 Dir 4.13	MEM30-C. Do not access freed memory
MISRA C:2012 Dir 4.14	INT04-C. Enforce limits on integer values originating from tainted sources
MISRA C:2012 Dir 4.14	INT10-C. Do not assume a positive remainder when using the % operator
MISRA C:2012 Dir 4.14	STR02-C. Sanitize data passed to complex subsystems
MISRA C:2012 Directive 4.14	EXP34-C. Do not dereference null pointers
MISRA C:2012 Rule 1.2	MSC04-C. Use comments consistently and in a readable fashion
MISRA C:2012 Rule 2.1	MSC07-C. Detect and remove dead code
MISRA C:2012 Rule 2.1	MSC12-C. Detect and remove code that has no effect or is never executed
MISRA C:2012 Rule 2.2	DCL22-C. Use volatile for data that cannot be cached
MISRA C:2012 Rule 2.2	MSC12-C. Detect and remove code that has no effect or is never executed
MISRA C:2012 Rule 3.1	MSC04-C. Use comments consistently and in a readable fashion
MISRA C:2012 Rule 4.2	PRE07-C. Avoid using repeated question marks
MISRA C:2012 Rule 5.1	DCL40-C. Do not create incompatible declarations of the same function or object
MISRA C:2012 Rule 5.1	DCL23-C. Guarantee that mutually visible identifiers are unique
MISRA C:2012 Rule 5.2	DCL23-C. Guarantee that mutually visible identifiers are unique
MISRA C:2012 Rule 5.3	DCL01-C. Do not reuse variable names in subscopes
MISRA C:2012 Rule 5.3	DCL23-C. Guarantee that mutually visible identifiers are unique
MISRA C:2012 Rule 5.4	DCL23-C. Guarantee that mutually visible identifiers are unique
MISRA C:2012 Rule 5.5	DCL23-C. Guarantee that mutually visible identifiers are unique
MISRA C:2012 Rule 6.1	INT12-C. Do not make assumptions about the type of a plain int bit-field when used in an expression
MISRA C:2012 Rule 7.1	DCL18-C. Do not begin integer constants with 0 when specifying a decimal value

MISRA C:2012 Rule 7.3	DCL16-C. Use "L," not "l," to indicate a long value
MISRA C:2012 Rule 8.1	DCL31-C. Declare identifiers before using them
MISRA C:2012 Rule 8.2	DCL36-C. Do not declare an identifier with conflicting linkage classifications
MISRA C:2012 Rule 8.2	DCL07-C. Include the appropriate type information in function declarators
MISRA C:2012 Rule 8.2	DCL20-C. Explicitly specify void when a function accepts no arguments
MISRA C:2012 Rule 8.3	DCL40-C. Do not create incompatible declarations of the same function or object
MISRA C:2012 Rule 8.3	EXP37-C. Call functions with the correct number and type of arguments
MISRA C:2012 Rule 8.4	DCL36-C. Do not declare an identifier with conflicting linkage classifications
MISRA C:2012 Rule 8.7	DCL15-C. Declare file-scope objects or functions that do not need external linkage as static
MISRA C:2012 Rule 8.7	DCL19-C. Minimize the scope of variables and functions
MISRA C:2012 Rule 8.8	DCL36-C. Do not declare an identifier with conflicting linkage classifications
MISRA C:2012 Rule 8.8	DCL15-C. Declare file-scope objects or functions that do not need external linkage as static
MISRA C:2012 Rule 8.9	DCL19-C. Minimize the scope of variables and functions
MISRA C:2012 Rule 8.11	ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer
MISRA C:2012 Rule 8.12	INT09-C. Ensure enumeration constants map to unique values
MISRA C:2012 Rule 8.13	DCL13-C. Declare function parameters that are pointers to values not changed by the function as const
MISRA C:2012 Rule 8.14	EXP43-C. Avoid undefined behavior when using restrict-qualified pointers
MISRA C:2012 Rule 9.5	ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer
MISRA C:2012 Rule 10.1	EXP46-C. Do not use a bitwise operator with a Boolean-like operand
MISRA C:2012 Rule 10.1	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
MISRA C:2012 Rule 10.1	INT02-C. Understand integer conversion rules
MISRA C:2012 Rule 10.1	INT07-C. Use only explicitly signed or unsigned char type for numeric values
MISRA C:2012 Rule 10.1	INT13-C. Use bitwise operators only on unsigned operands
MISRA C:2012 Rule 10.1	INT16-C. Do not make assumptions about representation of signed integers
MISRA C:2012 Rule 10.1	STR04-C. Use plain char for characters in the basic character set
MISRA C:2012 Rule 10.2	STR04-C. Use plain char for characters in the basic character set
MISRA C:2012 Rule 10.3	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
MISRA C:2012 Rule 10.3	FLP06-C. Convert integers to floating point for floating-point operations
MISRA C:2012 Rule 10.3	INT02-C. Understand integer conversion rules
MISRA C:2012 Rule 10.3	INT07-C. Use only explicitly signed or unsigned char type for numeric values
MISRA C:2012 Rule 10.3	STR04-C. Use plain char for characters in the basic character set
MISRA C:2012 Rule 10.4	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
MISRA C:2012 Rule 10.4	INT02-C. Understand integer conversion rules
MISRA C:2012 Rule 10.4	INT07-C. Use only explicitly signed or unsigned char type for numeric values
MISRA C:2012 Rule 10.4	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
MISRA C:2012 Rule 10.4	STR04-C. Use plain char for characters in the basic character set
MISRA C:2012 Rule 10.6	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
MISRA C:2012 Rule 10.6	INT02-C. Understand integer conversion rules
MISRA C:2012 Rule 10.6	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
MISRA C:2012 Rule 10.7	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
MISRA C:2012 Rule 10.7	INT02-C. Understand integer conversion rules
MISRA C:2012 Rule 10.7	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
MISRA C:2012 Rule 10.8	INT02-C. Understand integer conversion rules
MISRA C:2012 Rule 11.1	EXP36-C. Do not cast pointers into more strictly aligned pointer types
MISRA C:2012 Rule 11.1	EXP37-C. Call functions with the correct number and type of arguments
MISRA C:2012 Rule 11.1	DCL07-C. Include the appropriate type information in function declarators
MISRA C:2012 Rule 11.2	EXP36-C. Do not cast pointers into more strictly aligned pointer types

MISRA C:2012 Rule 11.3	EXP36-C. Do not cast pointers into more strictly aligned pointer types
MISRA C:2012 Rule 11.3	EXP39-C. Do not access a variable through a pointer of an incompatible type
MISRA C:2012 Rule 11.5	EXP36-C. Do not cast pointers into more strictly aligned pointer types
MISRA C:2012 Rule 11.6	INT36-C. Converting a pointer to integer or integer to pointer
MISRA C:2012 Rule 11.7	EXP36-C. Do not cast pointers into more strictly aligned pointer types
MISRA C:2012 Rule 11.8	EXP32-C. Do not access a volatile object through a nonvolatile reference
MISRA C:2012 Rule 11.8	EXP05-C. Do not cast away a const qualification
MISRA C:2012 Rule 12.1	EXP00-C. Use parentheses for precedence of operation
MISRA C:2012 Rule 12.5	ARR01-C. Do not apply the sizeof operator to a pointer when taking the size of an array
MISRA C:2012 Rule 13.2	PRE31-C. Avoid side effects in arguments to unsafe macros
MISRA C:2012 Rule 13.2	EXP30-C. Do not depend on the order of evaluation for side effects
MISRA C:2012 Rule 13.2	EXP10-C. Do not depend on the order of evaluation of subexpressions or the order in which side effects take place
MISRA C:2012 Rule 13.6	EXP44-C. Do not rely on side effects in operands to sizeof, _Alignof, or _Generic
MISRA C:2012 Rule 14.1	FLP30-C. Do not use floating-point variables as loop counters
MISRA C:2012 Rule 15.6	EXP19-C. Use braces for the body of an if, for, or while statement
MISRA C:2012 Rule 16.1	DCL41-C. Do not declare variables inside a switch statement before the first case label
MISRA C:2012 Rule 16.2	MSC20-C. Do not use a switch statement to transfer control into a complex block
MISRA C:2012 Rule 17.1	DCL10-C. Maintain the contract between the writer and caller of variadic functions
MISRA C:2012 Rule 17.1	DCL11-C. Understand the type issues associated with variadic functions
MISRA C:2012 Rule 17.1	ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy
MISRA C:2012 Rule 17.2	MEM05-C. Avoid large stack allocations
MISRA C:2012 Rule 17.3	DCL31-C. Declare identifiers before using them
MISRA C:2012 Rule 17.3	DCL36-C. Do not declare an identifier with conflicting linkage classifications
MISRA C:2012 Rule 17.3	EXP37-C. Call functions with the correct number and type of arguments
MISRA C:2012 Rule 17.7	ERR33-C. Detect and handle standard library errors
MISRA C:2012 Rule 18.1	ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
MISRA C:2012 Rule 18.1	ARR39-C. Do not add or subtract a scaled integer to a pointer
MISRA C:2012 Rule 18.1	EXP08-C. Ensure pointer arithmetic is used correctly
MISRA C:2012 Rule 18.2	ARR36-C. Do not subtract or compare two pointers that do not refer to the same array
MISRA C:2012 Rule 18.2	EXP08-C. Ensure pointer arithmetic is used correctly
MISRA C:2012 Rule 18.3	EXP08-C. Ensure pointer arithmetic is used correctly
MISRA C:2012 Rule 18.6	DCL30-C. Declare objects with appropriate storage durations
MISRA C:2012 Rule 18.6	MEM30-C. Do not access freed memory
MISRA C:2012 Rule 20.7	PRE01-C. Use parentheses within macros around parameter names
MISRA C:2012 Rule 21.1	DCL37-C. Do not declare or define a reserved identifier
MISRA C:2012 Rule 21.2	DCL37-C. Do not declare or define a reserved identifier
MISRA C:2012 Rule 21.3	API03-C. Create consistent interfaces and capabilities across related functions
MISRA C:2012 Rule 21.16	EXP42-C. Do not compare padding data
MISRA C:2012 Rule 21.17	STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string
MISRA C:2012 Rule 21.19	ENV30-C. Do not modify the object referenced by the return value of certain functions
MISRA C:2012 Rule 22.1	MEM30-C. Do not access freed memory
MISRA C:2012 Rule 22.2	MEM30-C. Do not access freed memory
MISRA C:2012 Rule 22.2	MEM34-C. Only free memory allocated dynamically
MISRA C:2012 Rule 22.7	FIO34-C. Distinguish between characters read from a file and EOF or WEOF
MISRA C:2012 Rule 22.8	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
MISRA C:2012 Rule 22.9	ERR33-C. Detect and handle standard library errors

MISRA C:2012 Rule 22.10	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
Missing break of switch case	MSC17-C. Finish every set of statements associated with a case label with a break statement
Missing byte reordering when transferring data	POS39-C. Use the correct byte ordering when transferring data between systems
Missing case for switch condition	MSC01-C. Strive for logical completeness
Missing case for switch condition	MSC07-C. Detect and remove dead code
Missing lock	CON01-C. Acquire and release synchronization primitives in the same module, at the same level of abstraction
Missing null in string array	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Missing null in string array	STR11-C. Do not specify the bound of a character array initialized with a string literal
Missing reset of a freed pointer	MEM01-C. Store a new value in pointers immediately after free()
Missing return statement	MSC37-C. Ensure that control never reaches the end of a non-void function
Missing unlock	CON01-C. Acquire and release synchronization primitives in the same module, at the same level of abstraction
Missing unlock	MEM12-C. Consider using a goto chain when leaving a function on error when using and releasing resources
Misuse of a FILE object	FIO38-C. Do not copy a FILE object
Misuse of errno	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
Misuse of errno in a signal handler	ERR32-C. Do not rely on indeterminate values of errno
Misuse of readlink()	POS30-C. Use the readlink() function properly
Misuse of return value from nonreentrant standard function	ENV34-C. Do not store pointers returned by certain functions
Misuse of sign-extended character value	STR34-C. Cast characters to unsigned char before converting to larger integer sizes
Misuse of sign-extended character value	STR37-C. Arguments to character-handling functions must be representable as an unsigned char
Misuse of structure with flexible-array member	MEM33-C. Allocate and copy structures containing a flexible array member dynamically
Modification of internal buffer returned from nonreentrant standard function	ENV30-C. Do not modify the object referenced by the return value of certain functions
Modification of internal buffer returned from nonreentrant standard function	STR06-C. Do not assume that strtok() leaves the parse string unchanged
Non-initialized pointer	EXP33-C. Do not read uninitialized memory
Non-initialized pointer	MSC15-C. Do not depend on undefined behavior
Non-initialized variable	EXP33-C. Do not read uninitialized memory
Non-initialized variable	MSC39-C. Do not call va_arg() on a va_list that has an indeterminate value
Non-initialized variable	MSC15-C. Do not depend on undefined behavior
Null pointer	EXP34-C. Do not dereference null pointers
Null pointer	MSC15-C. Do not depend on undefined behavior
Opening previously opened resource	FIO24-C. Do not open a file that is already open
Overlapping assignment	MSC15-C. Do not depend on undefined behavior
Pointer access out of bounds	EXP39-C. Do not access a variable through a pointer of an incompatible type
Pointer access out of bounds	ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
Pointer access out of bounds	ARR38-C. Guarantee that library functions do not form invalid pointers
Pointer access out of bounds	ARR39-C. Do not add or subtract a scaled integer to a pointer
Pointer access out of bounds	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Pointer access out of bounds	MEM35-C. Allocate sufficient memory for an object
Pointer access out of bounds	API02-C. Functions that read or write to or from an array should take an argument to specify the source or target size
Pointer access out of bounds	EXP08-C. Ensure pointer arithmetic is used correctly
Pointer access out of bounds	MSC15-C. Do not depend on undefined behavior
Pointer dereference with tainted offset	ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
Pointer dereference with tainted offset	API02-C. Functions that read or write to or from an array should take an argument to specify the source or target size

Pointer or reference to stack variable leaving scope	DCL30-C. Declare objects with appropriate storage durations
Possible misuse of sizeof	ARR38-C. Guarantee that library functions do not form invalid pointers
Possible misuse of sizeof	ARR39-C. Do not add or subtract a scaled integer to a pointer
Possible misuse of sizeof	ARR00-C. Understand how arrays work
Possible misuse of sizeof	ARR01-C. Do not apply the sizeof operator to a pointer when taking the size of an array
Possibly unintended evaluation of expression because of operator precedence rules	EXP00-C. Use parentheses for precedence of operation
Possibly unintended evaluation of expression because of operator precedence rules	EXP13-C. Treat relational and equality operators as if they were nonassociative
Predefined macro used as object	MSC38-C. Do not treat a predefined identifier as an object if it might only be implemented as a macro
Predictable block cipher initialization vector	MSC18-C. Be careful while handling sensitive data, such as passwords, in program code
Predictable cipher key	MSC18-C. Be careful while handling sensitive data, such as passwords, in program code
Predictable random output from predictable seed	MSC32-C. Properly seed pseudorandom number generators
Preprocessor directive in macro argument	PRE32-C. Do not use preprocessor directives in invocations of function-like macros
Privilege drop not verified	POS37-C. Ensure that privilege relinquishment is successful
Qualifier removed in conversion	EXP32-C. Do not access a volatile object through a nonvolatile reference
Qualifier removed in conversion	EXP37-C. Call functions with the correct number and type of arguments
Qualifier removed in conversion	EXP05-C. Do not cast away a const qualification
Resource leak	MEM12-C. Consider using a goto chain when leaving a function on error when using and releasing resources
Return from computational exception signal handler	SIG35-C. Do not return from a computational exception signal handler
Return value of a sensitive function not checked	ERR33-C. Detect and handle standard library errors
Returned value of a sensitive function not checked	POS54-C. Detect and handle POSIX library errors
Returned value of a sensitive function not checked	EXP12-C. Do not ignore values returned by functions
Sensitive data printed out	MEM06-C. Ensure that sensitive data is not written out to disk
Sensitive heap memory not cleared before release	MEM03-C. Clear sensitive information stored in reusable resources
Sensitive heap memory not cleared before release	MSC18-C. Be careful while handling sensitive data, such as passwords, in program code
Shared data access within signal handler	SIG31-C. Do not access shared objects in signal handlers
Shift of a negative value	INT34-C. Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand
Shift operation overflow	INT34-C. Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand
Side effect of expression ignored	EXP44-C. Do not rely on side effects in operands to sizeof, _Alignof, or _Generic
Sign change integer conversion overflow	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
Signal call from within signal handler	SIG34-C. Do not call signal() from within interruptible signal handlers
Standard function call with incorrect arguments	EXP37-C. Call functions with the correct number and type of arguments
Standard function call with incorrect arguments	STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string
Standard function call with incorrect arguments	FIO46-C. Do not access a closed file
Standard function call with incorrect arguments	API00-C. Functions should validate their parameters
Standard function call with incorrect arguments	MSC15-C. Do not depend on undefined behavior
Stream argument with possibly unintended side effects	FIO41-C. Do not call getc(), putc(), getwc(), or putwc() with a stream argument that has side effects
Subtraction or comparison between pointers to different arrays	ARR36-C. Do not subtract or compare two pointers that do not refer to the same array
Tainted Data Defects	API00-C. Functions should validate their parameters
Tainted division operand	INT32-C. Ensure that operations on signed integers do not result in overflow
Tainted division operand	INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors
Tainted modulo operand	INT32-C. Ensure that operations on signed integers do not result in overflow
Tainted modulo operand	INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors
Tainted modulo operand	INT10-C. Do not assume a positive remainder when using the % operator
Tainted NULL or non-null-terminated string	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator

Tainted NULL or non-null-terminated string	STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string
Tainted NULL or non-null-terminated string	ENV01-C. Do not make assumptions about the size of an environment variable
Tainted NULL or non-null-terminated string	FIO17-C. Do not rely on an ending null character when using fread()
Tainted sign change conversion	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
Tainted sign change conversion	INT02-C. Understand integer conversion rules
Tainted sign change conversion	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
Tainted sign change conversion	MEM04-C. Beware of zero-length allocations
Tainted sign change conversion	MEM11-C. Do not assume infinite heap space
Tainted sign change conversion	MSC21-C. Use robust loop termination conditions
Tainted size of variable length array	ARR32-C. Ensure size arguments for variable length arrays are in a valid range
Tainted size of variable length array	INT04-C. Enforce limits on integer values originating from tainted sources
Tainted size of variable length array	MEM04-C. Beware of zero-length allocations
Tainted size of variable length array	MEM05-C. Avoid large stack allocations
Tainted string format	FIO30-C. Exclude user input from format strings
Too many va_arg calls for current argument list	EXP47-C. Do not call va_arg with an argument of the incorrect type
Umask used with chmod-style arguments	FIO06-C. Create files with appropriate access permissions
Uncleared sensitive data in stack	MEM03-C. Clear sensitive information stored in reusable resources
Uncleared sensitive data in stack	MSC18-C. Be careful while handling sensitive data, such as passwords, in program code
Universal character name from token concatenation	PRE30-C. Do not create a universal character name through concatenation
Unprotected dynamic memory allocation	ERR33-C. Detect and handle standard library errors
Unprotected dynamic memory allocation	MEM10-C. Define and use a pointer validation function
Unprotected dynamic memory allocation	MEM11-C. Do not assume infinite heap space
Unreachable code	MSC01-C. Strive for logical completeness
Unreachable code	MSC07-C. Detect and remove dead code
Unreachable code	MSC12-C. Detect and remove code that has no effect or is never executed
Unreliable cast of function pointer	EXP37-C. Call functions with the correct number and type of arguments
Unreliable cast of function pointer	MSC15-C. Do not depend on undefined behavior
Unreliable cast of pointer	EXP36-C. Do not cast pointers into more strictly aligned pointer types
Unreliable cast of pointer	EXP39-C. Do not access a variable through a pointer of an incompatible type
Unreliable cast of pointer	STR38-C. Do not confuse narrow and wide character strings and functions
Unreliable cast of pointer	MSC15-C. Do not depend on undefined behavior
Unsafe call to a system function	ENV33-C. Do not call system()
Unsafe conversion between pointer and integer	INT36-C. Converting a pointer to integer or integer to pointer
Unsafe conversion from string to numeric value	ERR34-C. Detect errors when converting a string to a number
Unsafe standard encryption function	MSC18-C. Be careful while handling sensitive data, such as passwords, in program code
Unsigned integer conversion overflow	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data
Unsigned integer conversion overflow	FLP34-C. Ensure that floating-point conversions are within range of the new type
Unsigned integer conversion overflow	INT02-C. Understand integer conversion rules
Unsigned integer conversion overflow	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
Unsigned integer overflow	INT30-C. Ensure that unsigned integer operations do not wrap
Unused parameter	MSC13-C. Detect and remove unused values
Use of automatic variable as putenv-family function argument	POS34-C. Do not call putenv() with a pointer to an automatic variable as the argument
Use of dangerous standard function	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Use of dangerous standard function	API02-C. Functions that read or write to or from an array should take an argument to specify the source or target size
Use of dangerous standard function	ENV01-C. Do not make assumptions about the size of an environment variable
Use of dangerous standard function	PRE09-C. Do not replace secure functions with deprecated or obsolescent functions

Use of dangerous standard function	STR07-C. Use the bounds-checking interfaces for string manipulation
Use of indeterminate string	FIO40-C. Reset strings on fgets() or fgetws() failure
Use of memset with size argument zero	MSC12-C. Detect and remove code that has no effect or is never executed
Use of non-secure temporary file	FIO03-C. Do not make assumptions about fopen() and file creation
Use of non-secure temporary file	FIO21-C. Do not create temporary files in shared directories
Use of obsolete standard function	MSC33-C. Do not pass invalid data to the asctime() function
Use of obsolete standard function	POS33-C. Do not use vfork()
Use of obsolete standard function	MSC24-C. Do not use deprecated or obsolescent functions
Use of obsolete standard function	PRE09-C. Do not replace secure functions with deprecated or obsolescent functions
Use of plain char type for numerical value	INT07-C. Use only explicitly signed or unsigned char type for numeric values
Use of previously closed resource	FIO46-C. Do not access a closed file
Use of previously freed pointer	MEM30-C. Do not access freed memory
Use of previously freed pointer	MEM00-C. Allocate and free memory in the same module, at the same level of abstraction
Use of setjmp/longjmp	MSC22-C. Use the setjmp(), longjmp() facility securely
Use of tainted pointer	EXP34-C. Do not dereference null pointers
Use of tainted pointer	ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
Use of tainted pointer	ARR38-C. Guarantee that library functions do not form invalid pointers
Use of tainted pointer	API02-C. Functions that read or write to or from an array should take an argument to specify the source or target size
Use of tainted pointer	MEM10-C. Define and use a pointer validation function
Use of tainted pointer	MSC15-C. Do not depend on undefined behavior
Variable length array with nonpositive size	MEM04-C. Beware of zero-length allocations
Variable length array with nonpositive size	MEM05-C. Avoid large stack allocations
Variable shadowing	DCL01-C. Do not reuse variable names in subscopes
Vulnerable path manipulation	FIO02-C. Canonicalize path names originating from tainted sources
Vulnerable permission assignments	FIO06-C. Create files with appropriate access permissions
Vulnerable pseudo-random number generator	MSC30-C. Do not use the rand() function for generating pseudorandom numbers
Write without a further read	DCL22-C. Use volatile for data that cannot be cached
Write without a further read	MSC13-C. Detect and remove unused values
Writing to const qualified object	EXP40-C. Do not modify constant objects
Writing to const qualified object	STR30-C. Do not attempt to modify string literals
Writing to const qualified object	MSC15-C. Do not depend on undefined behavior
Writing to const qualified object	STR05-C. Use pointers to const when referring to string literals
Writing to const qualified object	STR06-C. Do not assume that strtok() leaves the parse string unchanged
Wrong allocated object size for cast	EXP36-C. Do not cast pointers into more strictly aligned pointer types
Wrong allocated object size for cast	STR38-C. Do not confuse narrow and wide character strings and functions
Wrong allocated object size for cast	MEM02-C. Immediately cast the result of a memory allocation function call into a pointer to the allocated type
Wrong type used in sizeof	MEM35-C. Allocate sufficient memory for an object
Wrong type used in sizeof	ARR00-C. Understand how arrays work
Wrong type used in sizeof	ARR01-C. Do not apply the sizeof operator to a pointer when taking the size of an array
Wrong type used in sizeof	MEM02-C. Immediately cast the result of a memory allocation function call into a pointer to the allocated type