Rose

Tool Version: unknown

Checker	Guideline
	DCL31-C. Declare identifiers before using them
	DCL37-C. Do not declare or define a reserved identifier
	EXP32-C. Do not access a volatile object through a nonvolatile reference
	EXP44-C. Do not rely on side effects in operands to sizeof, _Alignof, or _Generic
	INT36-C. Converting a pointer to integer or integer to pointer
	FLP30-C. Do not use floating-point variables as loop counters
	ARR37-C. Do not add or subtract an integer to a pointer to a non-array object
	ARR38-C. Guarantee that library functions do not form invalid pointers
	MEM30-C. Do not access freed memory
	MEM31-C. Free dynamically allocated memory when no longer needed
	FIO30-C. Exclude user input from format strings
	FIO34-C. Distinguish between characters read from a file and EOF or WEOF
	FIO42-C. Close files when they are no longer needed
	FIO46-C. Do not access a closed file
	ENV31-C. Do not rely on an environment pointer following an operation that may invalidate it
	ENV33-C. Do not call system()
	ENV34-C. Do not store pointers returned by certain functions
	MSC30-C. Do not use the rand() function for generating pseudorandom numbers
	POS30-C. Use the readlink() function properly
	POS33-C. Do not use vfork()

	POS34-C. Do not call putenv() with a pointer to an automatic variable as the argument
	ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer
	DCL00-C. Const-qualify immutable objects
	DCL01-C. Do not reuse variable names in subscopes
	DCL02-C. Use visually distinct identifiers
	DCL05-C. Use typedefs of non-pointer types only
	ENV02-C. Beware of multiple environment variables with the same effective name
	EXP05-C. Do not cast away a const qualification
	EXP12-C. Do not ignore values returned by functions
	EXP14-C. Beware of integer promotion when performing bitwise operations on integer types smaller than int
	FIO08-C. Take care when calling remove() on an open file
	FIO11-C. Take care when specifying the mode parameter of fopen()
	FIO22-C. Close files before spawning processes
	INT09-C. Ensure enumeration constants map to unique values
	INT12-C. Do not make assumptions about the type of a plain int bit-field when used in an expression
	MEM01-C. Store a new value in pointers immediately after free()
	MEM07-C. Ensure that the arguments to calloc(), when multiplied, do not wrap
	MSC17-C. Finish every set of statements associated with a case label with a break statement
	MSC21-C. Use robust loop termination conditions
	STR04-C. Use plain char for characters in the basic character set
	STR05-C. Use pointers to const when referring to string literals
	STR06-C. Do not assume that strtok() leaves the parse string unchanged
	STR11-C. Do not specify the bound of a character array initialized with a string literal
Automatically detects simple violations of this rule, although it may return some false positives. It may not catch more complex violations, such as initialization within functions taking uninitialized variables as arguments. It does catch the second noncompliant code example, and can be extended to catch the first as well	EXP33-C. Do not read uninitialized memory
A module written in Compass/ROSE can detect violations of this rule	CON33-C. Avoid race conditions when using library functions

Can catch violations of this rule by scanning the printf() and scanf() family of functions. For each such function, any variable that corresponds to a %d qualifier (or any qualifier besides %j) and that is not one of the built-in types (char, short, int, long, long long) indicates a violation of this rule. To catch violations, ROSE would also have to recognize derived types in expressions, such as size_t	INT15-C. Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types
Can detect all of these	MEM33-C. Allocate and copy structures containing a flexible array member dynamically
Can detect common violations of this rule. However, it cannot handle cases in which the value returned by fgetpos() is copied between several variables before being passed to fsetpos()	FIO44-C. Only use values for fsetpos() that are returned from fgetpos()
Can detect simple violations of this recommendation. In particular, it warns when two calls to ungetc() on the same stream are not interspersed with a file-positioning or file-read function. It cannot handle cases where ungetc() is called from inside a loop	FIO13-C. Never push back anything other than one read character
Can detect simple violations of this rule	STR30-C. Do not attempt to modify string literals
Can detect simple violations of this rule	FIO38-C. Do not copy a FILE object
Can detect simple violations of this rule	FIO39-C. Do not alternately input and output from a stream without an intervening flush or positioning call
Can detect simple violations of this rule. It needs to examine each expression and make sure that no variable is modified twice in the expression. It also must check that no variable is modified once, then read elsewhere, with the single exception that a variable may appear on both the left and right of an assignment operator	EXP30-C. Do not depend on the order of evaluation for side effects
Can detect some violations of this recommendation when checking EXP36-C. Do not cast pointers into more strictly aligned pointer types	MEM02-C. Immediately cast the result of a memory allocation function call into a pointer to the allocated type
Can detect some violations of this recommendation. In particular, it flags switch statements that do not have a default clause. ROSE should detect "fake switches" as well (that is, a chain of if statements each checking the value of the same variable). These if statements should always end in an else clause, or they should mathematically cover every possibility. For instance, consider the following: if (x > 0) { /* ... */ } else if (x < 0) { /* ... */ } else if (x == 0) { /* ... */ }	MSC01-C. Strive for logical completeness
Can detect some violations of this recommendation. In particular, it warns when chown(), stat(), or chmod() are called on an open file	FIO01-C. Be careful using functions that use file names for identification
Can detect some violations of this rule	STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string
Can detect some violations of this rule	MEM34-C. Only free memory allocated dynamically
Can detect some violations of this rule (In particular, it ensures that all operations involving division or modulo are preceded by a check ensuring that the second operand is nonzero.)	INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors
Can detect some violations of this rule but cannot flag violations involving universal names	DCL23-C. Guarantee that mutually visible identifiers are unique
Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro	ERR06-C. Understand the termination behavior of assert() and abort()
Can detect some violations of this rule. However, it does not flag implicit casts, only explicit ones	FLP34-C. Ensure that floating-point conversions are within range of the new type
Can detect some violations of this rule. In particular, it ensures that all calls to open() supply exactly two arguments if the second argument does not involve O_CREAT, and exactly three arguments if the second argument does involve O_CREAT	EXP37-C. Call functions with the correct number and type of arguments
Can detect some violations of this rule. In particular, it ensures that calls to open() that are preceded by a call to lstat() are also followed by a call to fstat().	POS35-C. Avoid race conditions while checking for the existence of a symbolic link
Can detect some violations of this rule. In particular, it warns if the last element of a struct is an array with a small index (0 or 1)	DCL38-C. Use the correct syntax when declaring a flexible array member
Can detect some violations of this rule. In particular, it warns when calls to setgid() are immediately preceded by a call to setuid()	POS36-C. Observe correct revocation order while relinquishing privileges
Can detect some violations of this rule. In particular, it warns when the argument to malloc() is a variable that has not been compared against 0 or that is known at compile time to be 0	MEM04-C. Beware of zero-length allocations

Can detect violations of the recommendation but cannot distinguish between incomplete array declarations and pointer declarations	ARR01-C. Do not apply the sizeof operator to a pointer when taking the size of an array
Can detect violations of the rule by using the same method as STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator	ENV01-C. Do not make assumptions about the size of an environment variable
Can detect violations of the rule for single-file programs	SIG30-C. Call only asynchronous-safe functions within signal handlers
Can detect violations of the rule. However, it is unable to handle cases involving strcpy_s() or manual string copies such as the one in the first example	STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator
Can detect violations of this recommendation	MSC05-C. Do not manipulate time_t typed values directly
Can detect violations of this recommendation by flagging invocations of the following functions:atoi()scanf(), fscanf(), sscanf() Others?	ERR34-C. Detect errors when converting a string to a number
Can detect violations of this recommendation when checking for violations of EXP12-C. Do not ignore values returned by functions and EXP34-C. Do not dereference null pointers	ERR33-C. Detect and handle standard library errors
Can detect violations of this recommendation when checking for violations of EXP12-C. Do not ignore values returned by functions and EXP34-C. Do not dereference null pointers	POS54-C. Detect and handle POSIX library errors
Can detect violations of this recommendation while checking for violations of recommendation DCL00-C. Const-qualify immutable objects	DCL13-C. Declare function parameters that are pointers to values not changed by the function as const
Can detect violations of this recommendation. However, it can detect only those violations where both bitwise and arithmetic operators are used in the same expression	INT14-C. Avoid performing bitwise and arithmetic operations on the same data
Can detect violations of this recommendation. In particular, it catches comparisons and operations where one operand is of type size_t or rsize_t and the other is not	INT01-C. Use rsize_t or size_t for all integer values representing the size of an object
Can detect violations of this recommendation. In particular, it checks to see if the arguments to an equality operator are of a floating-point type	FLP02-C. Avoid using floating-point numbers when precise computation is needed
Can detect violations of this recommendation. In particular, it flags any instance of a variable of type char (without a signed or unsigned qualifier) that appears in an arithmetic expression	INT07-C. Use only explicitly signed or unsigned char type for numeric values
Can detect violations of this recommendation. In particular, it looks for the size argument of malloc(), calloc(), or realloc() and flags when it does not find a sizeof operator in the argument expression. It does not flag if the return value is assigned to a char *; in this case a string is being allocated, and sizeof is unnecessary because sizeof(char) == 1	EXP09-C. Use sizeof to determine the size of a type or variable
Can detect violations of this recommendation. In particular, it notes uses of the scanf() family of functions where on the type specifier is a floating-point or integer type	INT05-C. Do not use input functions to convert character data if they cannot handle all possible inputs
Can detect violations of this recommendation. Specifically, it reports violations ifA pointer to one object is type cast to the pointer of a different objectThe pointed-to object of the (type cast) pointer is then modified arithmetically	EXP11-C. Do not make assumptions regarding the layout of structures with bit-fields
Can detect violations of this recommendation. Specifically, Rose reports use of tmpnam(), tmpnam_s(), tmpfile(), and mktemp()	FIO21-C. Do not create temporary files in shared directories
Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring validation, including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT_MAX, which itself requires no validation because it cannot wrap.)	INT30-C. Ensure that unsigned integer operations do not wrap
Can detect violations of this rule for single-file programs	SIG31-C. Do not access shared objects in signal handlers
Can detect violations of this rule when checking for violations of INT07-C. Use only explicitly signed or unsigned char type for numeric values	STR34-C. Cast characters to unsigned char before converting to larger integer sizes
Can detect violations of this rule. Any assignment operation where the type of the assiged-to value is float or double, but all the expressions to the right of the assignment are integral, is a violation of this rule	FLP06-C. Convert integers to floating point for floating-point operations
Can detect violations of this rule. However, false positives may occur on systems with persistent handlers	SIG34-C. Do not call signal() from within interruptible signal handlers
Can detect violations of this rule. However, false warnings may be raised if limits.h is included	INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data

Can detect violations of this rule. However, it does not flag explicit casts to void * and then back to another pointer type	EXP36-C. Do not cast pointers into more strictly aligned pointer types
Can detect violations of this rule. In particular, it ensures that all functions registered with atexit() do not call functions such as exit()	ENV32-C. All exit handlers must return normally
Can detect violations of this rule. In particular, it ensures that the result of getenv() is stored in a const variable	ENV30-C. Do not modify the object referenced by the return value of certain functions
Can detect violations of this rule. In particular, it flags bitwise operations that involved variables not declared with unsigned type	INT13-C. Use bitwise operators only on unsigned operands
Can detect violations of this rule. In particular, ROSE ensures that any pointer returned by malloc(), calloc(), or realloc() is first checked for NULL before being used (otherwise, it is free()-ed). ROSE does not handle cases where an allocation is assigned to an lvalue that is not a variable (such as a struct member or C++ function call returning a reference)	EXP34-C. Do not dereference null pointers
Can detect violations of this rule. It automatically detects returning pointers to local variables. Detecting more general cases, such as examples where static pointers are set to local variables which then go out of scope, would be difficult	DCL30-C. Declare objects with appropriate storage durations
Can detect violations of this rule. It should look for patterns of (a op1 b) op2 c wherec has a bigger type than a or bNeither a nor b is typecast to c's typeop2 is assignment or comparison	INT18-C. Evaluate integer expressions in a larger size before comparing or assigning to that size
Can detect violations of this rule. Unsigned operands are detected when checking for INT13-C. Use bitwise operators only on unsigned operands	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
Could be configured to catch violations of this rule. The way to catch the noncompliant code example is to first hunt for example code that follows this pattern: for (LPWSTR pwszTemp = pwszPath + 2; *pwszTemp != L'\\'; *pwszTemp++;) In particular, the iteration variable is a pointer, it gets incremented, and the loop condition does not set an upper bound on the pointer. Once this case is handled, ROSE can handle cases like the real noncompliant code example, which is effectively the same semantics, just different syntax	ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
Could catch violations of this rule by enforcing that any call to open() or fopen() is preceded by a canonicalization routine—that is, a call to realpath() or canonicalize_file_name(). This call will catch some false positives, as ROSE cannot tell when canonicalization is warranted. False positives can be reduced (but not eliminated) by only reporting instances of fopen() or open() where the file name string has some other processing done to it. This reflects the fact that canonicalization is only necessary for doing verification based on the file name string	FIO02-C. Canonicalize path names originating from tainted sources
Could check violations of this rule by examining the size expression to malloc() or memcpy() functions. Specifically, the size argument should be bounded by 0, SIZE_MAX, and, unless it is a variable of type size_t or rsize_t, it should be bounds-checked before the malloc() call. If the argument is of the expression a*b, then an appropriate check is if (a < SIZE_MAX / b && a > 0) ...	MEM35-C. Allocate sufficient memory for an object
Could detect possible violations by flagging any signal handler that calls signal() to (re)assert itself as the handler for its signal	SIG01-C. Understand implementation-specific details regarding signal handler persistence
Could detect possible violations by reporting any function that has malloc() or free() but not both. This would catch some false positives, as there would be no way to tell if malloc() and free() are at the same level of abstraction if they are in different functions	MEM00-C. Allocate and free memory in the same module, at the same level of abstraction
Could detect possible violations of this recommendation by reporting expressions with side effects, including function calls, that appear on the right-hand side of an && or || operator	EXP02-C. Be aware of the short-circuit behavior of the logical AND and OR operators
Could detect possible violations of this rule by first flagging any usage of realloc(). Also, it could flag any usage of free that is not preceded by code to clear out the preceding memory, using memset. This heuristic is imperfect because it flags all possible data leaks, not just leaks of "sensitive" data, because ROSE cannot tell which data is sensitive	MEM03-C. Clear sensitive information stored in reusable resources
Could detect some violations of this rule (In particular, it could detect the noncompliant code example by searching for fgets(), followed by strlen() - 1, which could be 1. The crux of this rule is that a string returned by fgets() could still be empty, because the first char is '\0'. There are probably other code examples that violate this guideline; they would need to be enumerated before ROSE could detect them.)	FIO37-C. Do not assume that fgets() or fgetws() returns a nonempty string when successful
Could detect some violations of this rule. This rule applies only to untrusted file name strings, and ROSE cannot tell which strings are trusted and which are not. The best heuristic is to note if there is any verification of the file name before or after the fopen() call. If there is any verification, then the file opening should be preceded by an lstat() call and succeeded by an fstat() call. Although that does not enforce the rule completely, it does indicate that the coder is aware of the lstat-fopen-fstat idiom	FIO32-C. Do not perform operations on devices that are only appropriate for files
Could detect the specific noncompliant code example. It could identify when the result of a % operation might be negative and flag usage of that result in an array index. It could conceivably flag usage of any such result without first checking that the result is positive, but it would likely introduce many false positives	INT10-C. Do not assume a positive remainder when using the % operator
Could detect violations in the following manner: all calls to strncpy() and the other functions should be followed by an assignment of a terminating character to null-terminate the string	STR03-C. Do not inadvertently truncate a string
Could detect violations of this recommendation by flagging any comparison expression involving addition that could potentially overflow. For example, instead of comparing a + b < c (where b and c are compile-time constants) and b > c, the code should compare a < c - b. (This assumes a, b, c are unsigned ints. Usually b is small and c is an upper bound such as INT_MAX.)	INT08-C. Verify that all integer values are in range
Could detect violations of this recommendation by identifying any assignment expression as the top-level expression in an if or while statement	EXP45-C. Do not perform assignments in selection statements

Could detect violations of this recommendation by searching for the following pattern:Any expression that calls two functions between the same sequence pointsThose two functions both modify the value of a static variableThat static variable's value is referenced by code following the expression	EXP10-C. Do not depend on the order of evaluation of subexpressions or the order in which side effects take place
Could detect violations of this recommendation merely by searching for the use of "magic numbers" and magic strings in the code itself. That is, any number (except a few canonical numbers: 1, 0, 1, 2) that appears in the code anywhere besides where assigned to a variable is a magic number and should instead be assigned to a const integer, enum, or macro. Likewise, any string literal (except "" and individual characters) that appears in the code anywhere besides where assigned to a char* or char[] is a magic string	DCL06-C. Use meaningful symbolic constants to represent literal values
Could detect violations of this rule by ensuring that each library function is accompanied by the proper treatment 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
Could detect violations of this rule by ensuring that floating-point operations are surrounded by feclearexcept() and fetestexcept(). It would need to look for type conversions to float or double, divisions (by a number not known to be nonzero), and multiplication. It may be wisest to apply this to all floating-point operations in general	FLP03-C. Detect and handle floating-point errors
Could detect violations of this rule by looking for signal handlers that themselves call signal(). A violation is reported if the call fails and the handler therefore checks errno. A violation also exists if the signal handler modifies errno without first copying its value elsewhere	ERR32-C. Do not rely on indeterminate values of errno
Could detect violations of this rule by seeing if the argument to a character handling function (listed above) is not an unsigned char	STR37-C. Arguments to character-handling functions must be representable as an unsigned char
Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation	DCL03-C. Use a static assertion to test the value of a constant expression
Could detect violations of this rule merely by reporting functions that call abort(), exit(), or _Exit() inside an if or switch statement. This would also catch many false positives, as ROSE could not distinguish a library function from an application function	ERR05-C. Application-independent code should provide error detection without dictating error handling
Could flag possible violations of this rule by noting any pointer to struct that is passed to fread(), as the noncompliant code example demonstrates	FIO09-C. Be careful with binary data when transferring data across systems
Could report possible violations of this rule by flagging calls to open() that do not have an O_NOFOLLOW flag and that are not preceded by a call to lstat() and succeeded by a call to fstat	POS01-C. Check for the existence of links when dealing with files
Could report possible violations of this rule merely by reporting any open() or fopen() call that did not have a subsequent call to fstat()	FIO05-C. Identify files using multiple file attributes
Does not currently detect violations of this recommendation. Although the recommendation in general cannot be automated, because of the difficulty in enforcing contracts between a variadic function and its invokers, it would be fairly easy to enforce type correctness on arguments to the printf() family of functions	DCL11-C. Understand the type issues associated with variadic functions