Coverity Support for MISRA Coding Standards

Fully ensure the safety, reliability, and security of software written in C and C++

Overview

Software is eating the world. Industries that have traditionally relied on mechanical, electronic, and analog control systems are increasingly replacing them with software-driven systems. For example, the average car is expected to contain 300 million lines of code in the next decade—up from 100 million lines of code today. But with the growth of software comes the growth of software defects—which can manifest themselves in these systems with tangible and life-threatening consequences. More importantly, malicious actors can deliberately trigger failures for their own purposes.

Highly complex industries involve many vendors and suppliers simultaneously contributing to the software that goes into the final product. Every participant in this software supply chain must reach consensus on coding standards, defect reporting structures, and so on. The confluence of these two factors—increasing complexity in software systems and longer software supply chains—creates the demand for new tools.

The MISRA C and C++ coding standards are widely used in safety-critical industries, such as automotive, medical, military, and aerospace. The standards provide a set of best practices for writing C and C++ code, facilitating the authorship of safe, secure, and portable code. With Coverity® static analysis, Synopsys provides a comprehensive solution for MISRA standard compliance that is scalable from individual developers all the way to complex software supply chains.

ISO/IEC 9899:2011. Coverity covers the entire MISRA C:2012 standard, including Amendments 1 and 2.*

* All rules that can be checked by static analysis are supported. MISRA C:2012 contains 6 rules that are not statically checkable, so a total of 169 are implemented (out of 175)

MISRA C:2012 rule coverage

	Decidable		Undecidable		Subtotal		Percent coverage
	Supported	All	Supported	All	Supported	All	
All	121	121	48	54	169	175	96.6%
Mandatory	5	5	11	11	16	16	100.0%
Required	88	88	27	32	113	118	95.8%
Advisory	28	28	10	11	38	39	97.4%

MISRA C:2012

The MISRA C:2012 coding standard supports the C90 and C99 language specifications. MISRA C:2012 Amendment 1 was released in 2016 and consists of 173 guidelines: 156 rules and 17 directives. In addition, checker implementations adhere to the Technical Corrigendum 1, released in July 2017.

Amendment 2 was announced in 2020 and adds 2 new rules, making it to a total of 175 guidelines: 158 rules and 17 directives. The two new rules in Amendment 2 introduce the support for ISO/IEC 9899:2011. Coverity covers the entire MISRA C:2012 standard, including Amendments 1 and 2.

MISRA C:2012 rules

Rule	Rule name	Category	Decidability	Supported	Notes
Directive 1.1	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood	Required	Undecidable	No	This directive is not statically verifiable.
Directive 2.1	All source files shall compile without any compilation errors	Required	Undecidable	No	No checker, but a successful analysis run confirms compliance.
Directive 3.1	All code shall be traceable to documented requirements	Required	Undecidable	No	This directive is not statically verifiable.
Directive 4.1	Run-time failures shall be minimized	Required	Undecidable	No	No checker, but the use of MISRA analysis will assist in minimizing runtime failures.
Directive 4.2	All usage of assembly language should be documented	Advisory	Undecidable	No	This directive is not statically verifiable.
Directive 4.3	Assembly language shall be encapsulated and isolated	Required	Undecidable	Yes	
Directive 4.4	Sections of code should not be “commented out”	Advisory	Undecidable	Yes	
Directive 4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous	Advisory	Undecidable	Yes	
Directive 4.6	*typedefs* that indicate size and signedness should be used in place of the basic numerical types	Advisory	Undecidable	Yes	Adheres to Technical Corrigendum 1.
Directive 4.7	If a function returns error information, then that error information shall be tested	Required	Undecidable	Yes	
Directive 4.8	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Advisory	Undecidable	Yes	Adheres to Technical Corrigendum 1.
Directive 4.9	A function should be used in preference to a *function-like macro* where they are interchangeable	Advisory	Undecidable	Yes	

Rule	Rule name	Category	Decidability	Supported	Notes
Directive 4.10	Precautions shall be taken in order to prevent the contents of a *header file* being included more than once	Required	Undecidable	Yes	
Directive 4.11	The validity of values passed to library functions shall be checked	Required	Undecidable	Yes	Adheres to Technical Corrigendum 1.
Directive 4.12	Dynamic memory allocation shall not be used	Required	Undecidable	Yes	
Directive 4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence	Advisory	Undecidable	Yes	
Directive 4.14	The validity of values received from external sources shall be checked	Required	Undecidable	Yes	New directive in Amendment 1.
Rule 1.1	The program shall contain no violations of the standard C syntax and *constraints*, and shall not exceed the implementation's translation limits	Required	Decidable	Yes	No checker, but a successful analysis run confirms compliance.
Rule 1.2	Language extensions should not be used	Advisory	Undecidable	Yes	
Rule 1.3	There shall be no occurrence of undefined or critical unspecified behaviour	Required	Undecidable	No	
Rule 1.4	Emergent language features shall not be used	Required	Decidable	Yes	New rule in Amendment 2.
Rule 2.1	A project shall not contain *unreachable code*	Required	Undecidable	Yes	
Rule 2.2	There shall be no *dead code*	Required	Undecidable	Yes	Adheres to Technical Corrigendum 1.
Rule 2.3	A project should not contain unused type declarations	Advisory	Decidable	Yes	
Rule 2.4	A project should not contain unused tag declarations	Advisory	Decidable	Yes	
Rule 2.5	A project should not contain unused macro declarations	Advisory	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 2.6	A function should not contain unused label declarations	Advisory	Decidable	Yes	
Rule 2.7	There should be no unused parameters in functions	Advisory	Decidable	Yes	
Rule 3.1	The character sequences /* and // shall not be used within a comment	Required	Decidable	Yes	
Rule 3.2	Line-splicing shall not be used in // comments	Required	Decidable	Yes	
Rule 4.1	Octal and hexadecimal escape sequences shall be terminated	Required	Decidable	Yes	
Rule 4.2	Trigraphs should not be used	Advisory	Decidable	Yes	
Rule 5.1	*External identifiers* shall be distinct	Required	Decidable	Yes	

Rule 5.2	Identifiers declared in the same *scope* and name space shall be distinct	Required	Decidable	Yes	
Rule 5.3	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope	Required	Decidable	Yes	
Rule 5.4	*Macro identifiers* shall be distinct	Required	Decidable	Yes	
Rule 5.5	Identifiers shall be distinct from macro names	Required	Decidable	Yes	
Rule 5.6	A *typedef* name shall be a unique identifier	Required	Decidable	Yes	
Rule 5.7	A tag name shall be a unique identifier	Required	Decidable	Yes	
Rule 5.8	Identifiers that define objects or functions with external linkage shall be unique	Required	Decidable	Yes	

Rule	Rule name	Category	Decidability	Supported	Notes
Rule 5.9	Identifiers that define objects or functions with internal linkage should be unique	Advisory	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 6.1	Bit-fields shall only be declared with an appropriate type	Required	Decidable	Yes	
Rule 6.2	Single-bit named bit fields shall not be of a signed type	Required	Decidable	Yes	
Rule 7.1	Octal constants shall not be used	Required	Decidable	Yes	
Rule 7.2	A “u” or “U” suffix shall be applied to all integer constants that are represented in an unsigned type	Required	Decidable	Yes	
Rule 7.3	The lowercase character “l” shall not be used in a literal suffix	Required	Decidable	Yes	
Rule 7.4	A string literal shall not be *assigned* to an object unless the object’s type is “pointer to *const*qualified *char”*	Required	Decidable	Yes	This rule was updated in Amendment 2.
Rule 8.1	Types shall be explicitly specified	Required	Decidable	Yes	
Rule 8.2	Function types shall be in *prototype form* with named parameters	Required	Decidable	Yes	
Rule 8.3	All declarations of an object or function shall use the same names and type qualifiers	Required	Decidable	Yes	
Rule 8.4	A compatible declaration shall be visible when an object or function with external linkage is defined	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 8.5	An external object or function shall be declared once in one and only one file	Required	Decidable	Yes	
Rule 8.6	An identifier with external linkage shall have exactly one external definition	Required	Decidable	Yes	

Rule	Rule name	Category	Decidability	Supported	Notes
Rule 8.7	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit	Advisory	Decidable	Yes	
Rule 8.8	The *static* storage class specifier shall be used in all declarations of objects and functions that have internal linkage	Required	Decidable	Yes	
Rule 8.9	An object should be defined at block scope if its identifier only appears in a single function	Advisory	Decidable	Yes	
Rule 8.10	An *inline function* shall be declared with the static storage class	Required	Decidable	Yes	
Rule 8.11	When an array with external linkage is declared, its size should be explicitly specified	Advisory	Decidable	Yes	
Rule 8.12	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Required	Decidable	Yes	
Rule 8.13	A pointer should point to a *const*-qualified type whenever possible	Advisory	Undecidable	Yes	
Rule 8.14	The *restrict* type qualifier shall not be used	Required	Decidable	Yes	
Rule 9.1	The value of an object with automatic storage duration shall not be read before it has been set	Mandatory	Undecidable	Yes	
Rule 9.2	The initializer for an aggregate or union shall be enclosed in braces	Required	Decidable	Yes	
Rule 9.3	Arrays shall not be partially initialized	Required	Decidable	Yes	
Rule 9.4	An element of an object shall not be in initialized more than once	Required	Decidable	Yes	
Rule 9.5	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Required	Decidable	Yes	
Rule 10.1	Operands shall not be of an inappropriate *essential type*	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 10.2	Expressions of *essentially character type* shall not be used inappropriately in addition and subtraction operations	Required	Decidable	Yes	
Rule 10.3	The value of an expression shall not be assigned to an object with a narrower *essential type* or of a different *essential type* category	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 10.4	Both operands of an operator in which the *usual arithmetic conversions* are performed shall have the same *essential type category*	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 10.5	The value of an expression should not be cast to an inappropriate *essential type*	Advisory	Decidable	Yes	Adheres to Technical Corrigendum 1.

Rule	Rule name	Category	Decidability	Supported	Notes
Rule 10.6	The value of a *composite expression* shall not be assigned to an object with wider *essential type*	Required	Decidable	Yes	
Rule 10.7	If a *composite expression* is used as one operand of an operator in which the *usual arithmetic conversions* are performed then the other operand shall not have wider *essential type*	Required	Decidable	Yes	
Rule 10.8	The value of a *composite expression* shall not be cast to a different *essential type category* or a wider *essential type*	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 11.1	Conversions shall not be performed between a pointer to a function and any other type	Required	Decidable	Yes	
Rule 11.2	Conversions shall not be performed between a pointer to an incomplete type and any other type	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type	Required	Decidable	Yes	
Rule 11.4	A conversion should not be performed between a pointer to object and an integer type	Advisory	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 11.5	A conversion should not be performed from pointer to *void* into pointer to object	Advisory	Decidable	Yes	
Rule 11.6	A cast shall not be performed between pointer to *void* and an arithmetic type	Required	Decidable	Yes	
Rule 11.7	A cast shall not be performed between pointer to object and a non-integer arithmetic type	Required	Decidable	Yes	
Rule 11.8	A cast shall not remove any *const* or *volatile* qualification from the type pointed to by a pointer	Required	Decidable	Yes	
Rule 11.9	The macro `NULL` shall be the only permitted form of integer *null pointer constant*	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 12.1	The precedence of operators within expressions should be made explicit	Advisory	Decidable	Yes	This rule was updated in Amendment 2.
Rule 12.2	The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the *essential type* of the left hand operand	Required	Undecidable	Yes	
Rule 12.3	The comma operator should not be used	Advisory	Decidable	Yes	
Rule 12.4	Evaluation of *constant expressions* should not lead to unsigned integer wrap-around	Advisory	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 12.5	The *sizeof* operator shall not have an operand which is a function parameter declared as “array of type”	Mandatory	Decidable	Yes	New rule in Amendment 1.
Rule 13.1	*Initializer lists* shall not contain *persistent side effects*	Required	Undecidable	Yes	

Rule 13.2	The value of an expression and its *persistent side effects* shall be the same under all permitted evaluation orders	Required	Undecidable	Yes	Adheres to Technical Corrigendum 1.
Rule 13.3	A full expression containing an increment (++) or decrement (--) operator should have no other potential *side effects* other than that caused by the increment or decrement operator	Advisory	Decidable	Yes	
Rule 13.4	The result of an assignment operator should not be *used*	Advisory	Decidable	Yes	
Rule 13.5	The right hand operand of a logical `&&` or `||` operator shall not contain *persistent side effects*	Required	Undecidable	Yes	
Rule 13.6	The operand of the *sizeof* operator shall not contain any expression which has potential *side effects*	Mandatory	Decidable	Yes	
Rule 14.1	A *loop counter* shall not have *essentially floating* type	Required	Undecidable	Yes	
Rule 14.2	A *for* loop shall be well-formed	Required	Undecidable	Yes	Adheres to Technical Corrigendum 1.
Rule 14.3	Controlling expressions shall not be invariant	Required	Undecidable	Yes	
Rule 14.4	The controlling expression of an *if* statement and the controlling expression of an *iteration-statement* shall have *essentially Boolean* type	Required	Decidable	Yes	
Rule 15.1	The *goto* statement should not be used	Advisory	Decidable	Yes	
Rule 15.2	The *goto* statement shall jump to a label declared later in the same function	Required	Decidable	Yes	
Rule 15.3	Any label referenced by a *goto* statement shall be declared in the same block, or in any block enclosing the *goto* statement	Required	Decidable	Yes	
Rule 15.4	There should be no more than one *break* or *goto* statement used to terminate any iteration statement	Advisory	Decidable	Yes	
Rule 15.5	A function should have a single point of exit at the end	Advisory	Decidable	Yes	
Rule 15.6	The body of an *iteration-statement* or a *selection-statement* shall be a *compound-statement*	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 15.7	All *if ... else if* constructs shall be terminated with an *else* statement	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 16.1	All *switch* statements shall be well-formed	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 16.2	A *switch label* shall only be used when the most closely-enclosing compound statement is the body of a *switch* statement	Required	Decidable	Yes	

Rule	Rule name	Category	Decidability	Supported	Notes
Rule 16.3	An unconditional *break* statement shall terminate every *switch-clause*	Required	Decidable	Yes	
Rule 16.4	Every *switch* statement shall have a *default* label	Required	Decidable	Yes	
Rule 16.5	A *default* label shall appear as either the first or the last *switch label* of a *switch* statement	Required	Decidable	Yes	
Rule 16.6	Every *switch* statement shall have at least two *switch-clauses*	Required	Decidable	Yes	
Rule 16.7	A *switch-expression* shall not have *essentially Boolean type*	Required	Decidable	Yes	
Rule 17.1	The features of `<stdarg.h>` shall not be used	Required	Decidable	Yes	
Rule 17.2	Functions shall not call themselves, either directly or indirectly	Required	Undecidable	Yes	
Rule 17.3	A function shall not be declared implicitly	Mandatory	Decidable	Yes	
Rule 17.4	All exit paths from a function with non-*void* return type shall have an explicit *return* statement with an expression	Mandatory	Decidable	Yes	
Rule 17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements	Advisory	Undecidable	Yes	
Rule 17.6	The declaration of an array parameter shall not contain the *static* keyword between the `[ ]`	Mandatory	Decidable	Yes	
Rule 17.7	The value returned by a function having non-*void* return type shall be *used*	Required	Decidable	Yes	
Rule 17.8	A function parameter should not be modified	Advisory	Undecidable	Yes	
Rule 18.1	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand	Required	Undecidable	Yes	
Rule 18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array	Required	Undecidable	Yes	
Rule 18.3	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object	Required	Undecidable	Yes	
Rule 18.4	The +, -, += and -= operators should not be applied to an expression of pointer type	Advisory	Decidable	Yes	
Rule 18.5	Declarations should contain no more than two levels of pointer nesting	Advisory	Decidable	Yes	
Rule 18.6	The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist	Required	Undecidable	Yes	
Rule 18.7	Flexible array members shall not be declared	Required	Decidable	Yes	

Rule	Rule name	Category	Decidability	Supported	Notes
Rule 18.8	Variable-length array types shall not be used	Required	Decidable	Yes	
Rule 19.1	An object shall not be assigned or copied to an overlapping object	Mandatory	Undecidable	Yes	Adheres to Technical Corrigendum 1.
Rule 19.2	The *union* keyword should not be used	Advisory	Decidable	Yes	
Rule 20.1	*#include* directives should only be preceded by preprocessor directives or comments	Advisory	Decidable	Yes	
Rule 20.2	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name	Required	Decidable	Yes	
Rule 20.3	The *#include* directive shall be followed by either a `<filename>` or `"filename"` sequence	Required	Decidable	Yes	
Rule 20.4	A macro shall not be defined with the same name as a keyword	Required	Decidable	Yes	
Rule 20.5	*#undef* should not be used	Advisory	Decidable	Yes	
Rule 20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument	Required	Decidable	Yes	
Rule 20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses	Required	Decidable	Yes	
Rule 20.8	The controlling expression of a *#if* or *#elif* preprocessing directive shall evaluate to 0 or 1	Required	Decidable	Yes	
Rule 20.9	All identifiers used in the controlling expression of *#if* or *#elif* preprocessing directives shall be *#define*'d before evaluation	Required	Decidable	Yes	
Rule 20.10	The # and ## preprocessor operators should not be used	Advisory	Decidable	Yes	
Rule 20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	Required	Decidable	Yes	
Rule 20.12	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	Required	Decidable	Yes	
Rule 20.13	A line whose first token is # shall be a valid preprocessing directive	Required	Decidable	Yes	
Rule 20.14	All *#else, #elif* and *#endif* preprocessor directives shall reside in the same file as the *#if, #ifdef* or *#ifndef* directive to which they are related	Required	Decidable	Yes	
Rule 21.1	*#define* and *#undef* shall not be used on a reserved identifier or reserved macro name	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 21.2	A reserved identifier or macro name shall not be declared	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.

Rule	Rule name	Category	Decidability	Supported	Notes
Rule 21.3	The memory allocation and deallocation functions of `<stdlib.h>` shall not be used	Required	Decidable	Yes	This rule was updated in Amendment 2.
Rule 21.4	The standard *header file*`<setjmp.h>`shall not be used	Required	Decidable	Yes	
Rule 21.5	The standard *header file* `<signal.h>` shall not be used	Required	Decidable	Yes	
Rule 21.6	The Standard Library input/output functions shall not be used	Required	Decidable	Yes	
Rule 21.7	The *atof, atoi, atol* and *atoll* functions of `<stdlib.h>` shall not be used	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 21.8	The standard library termination functions of `<stdlib.h>` shall not be used	Required	Decidable	Yes	Changed in Amendment 1 (removed "getenv"). See also Rule 21.19 and Rule 21.20. Adheres to Technical Corrigendum 1. Further updated in Amendment 2 and definition changed to include termination functions of <stdlib.h>.
Rule 21.9	The library functions *bsearch* and *qsort* of `<stdlib.h>` shall not be used	Required	Decidable	Yes	Adheres to Technical Corrigendum 1.
Rule 21.10	The Standard Library time and date functions shall not be used	Required	Decidable	Yes	This rule was updated in Amendment 2.
Rule 21.11	The standard *header file* `<tgmath.h>` shall not be used	Required	Decidable	Yes	
Rule 21.12	The exception handling features of `<fenv.h>` should not be used	Advisory	Decidable	Yes	
Rule 21.13	Any value passed to a function in `<ctype.h>` shall be representable as an *unsigned* char or be the value `EOF`	Mandatory	Undecidable	Yes	New rule in Amendment 1.
Rule 21.14	The Standard Library function *memcmp* shall not be used to compare null terminated strings	Required	Undecidable	Yes	New rule in Amendment 1.
Rule 21.15	The pointer arguments to the Standard Library functions *memcpy, memmove* and *memcmp* shall be pointers to qualified or unqualified versions of compatible types	Required	Decidable	Yes	New rule in Amendment 1.
Rule 21.16	The pointer arguments to the Standard Library function *memcmp* shallpoint to either a pointer type, an *essentially signed* type, an *essentially unsigned* type, an *essentially Boolean* type or an *essentially* enum type	Required	Decidable	Yes	New rule in Amendment 1.

Rule	Rule name	Category	Decidability	Supported	Notes
Rule 22.6	The value of a pointer to a `FILE` shall not be used after the associated stream has been closed	Mandatory	Undecidable	Yes	
Rule 22.7	The macro `EOF` shall only be compared with the unmodified return value from any Standard Library function capable of returning `EOF`	Required	Undecidable	Yes	New rule in Amendment 1.
Rule 22.8	The value of `errno` shall be set to zero prior to a call to an *errno-setting-function*	Required	Undecidable	Yes	New rule in Amendment 1.
Rule 22.9	The value of `errno` shall be tested against zero after calling an *errno-setting-function*	Required	Undecidable	Yes	New rule in Amendment 1.
Rule 22.10	The value of `errno` shall only be tested when the last function to be called was an *errno-setting-function*	Required	Undecidable	Yes	New rule in Amendment 1.
Rule 21.17	Use of the string handling functions from `<string.h>` shall not resultin accesses beyond the bounds of the objects referenced by their pointer parameters	Mandatory	Undecidable	Yes	New rule in Amendment 1.
Rule 21.18	The `size_t` argument passed to any function in `<string.h>` shall have an appropriate value	Mandatory	Undecidable	Yes	New rule in Amendment 1.
Rule 21.19	The pointers returned by the Standard Library functions *localeconv, getenv, setlocale* or, *strerror* shall only be used as if they have pointer to const-qualified type	Mandatory	Undecidable	Yes	New rule in Amendment 1.
Rule 21.20	The pointer returned by the Standard Library functions *asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale* or *strerror* shall not be used following a subsequent call to the same function	Mandatory	Undecidable	Yes	New rule in Amendment 1.
Rule 21.21	The standard library function system of `<stdlib.h>` shall not be used	Required	Decidable	Yes	New rule in Amendment 2.
Rule 22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released	Required	Undecidable	Yes	This rule was updated in Amendment 2.
Rule 22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function	Mandatory	Undecidable	Yes	
Rule 22.3	The same file shall not be open for read and write access at the same time on different streams	Required	Undecidable	Yes	
Rule 22.4	There shall be no attempt to write to a stream which has been opened as read-only	Mandatory	Undecidable	Yes	
Rule 22.5	A pointer to a `FILE` object shall not be dereferenced	Mandatory	Undecidable	Yes	

This datasheet applies to Coverity 2021.03 and later versions.

The Synopsys difference

Synopsys helps development teams build secure, high-quality software, minimizing risks while maximizing speed and productivity. Synopsys, a recognized leader in application security, provides static analysis, software composition analysis, and dynamic analysis solutions that enable teams to quickly find and fix vulnerabilities and defects in proprietary code, open source components, and application behavior.

For more information about the Synopsys Software Integrity Group, visit us online at www.synopsys.com/software.

Synopsys, Inc.
185 Berry Street, Suite 6500
San Francisco, CA 94107 USA

U.S. Sales: 800.873.8193
International Sales: +1 415.321.5237
Email: sig-info@synopsys.com