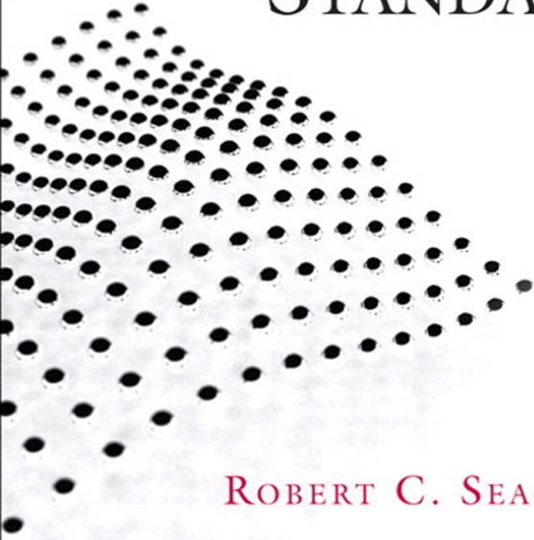
SEI SERIES • A CERT® BOOK

The CERT® C Secure Coding Standard



Robert C. Seacord

The CERT® C Secure Coding Standard

The CERT C Secure Coding Standard

Table of Contents

Contents

Preface

Acknowledgments

About the Author

Chapter 1 Using This Standard

- System Qualities
- Automatically Generated Code
- Compliance

Chapter 2 Preprocessor (PRE)

- PRE00-C. Prefer inline or static functions to function-like macros
- PRE01-C. Use parentheses within macros around parameter names
- PRE02-C. Macro replacement lists should be parenthesized
- PRE03-C. Prefer type definitions to defines for encoding types
- PRE04-C. Do not reuse a standard header file name
- PRE05-C. Understand macro replacement when concatenating tokens or performing stringification
- PRE06-C. Enclose header files in an inclusion guard
- PRE07-C. Avoid using repeated question marks
- PRE08-C. Guarantee that header file names are unique
- PRE09-C. Do not replace secure functions with less secure functions
- PRE10-C. Wrap multistatement macros in a do-while loop
- PRE30-C. Do not create a universal character name through concatenation

Table of Contents

PRE31-C. Never invoke an unsafe macro with arguments containing assignment, increment, decrement, volatile access, or function call

Chapter 3 Declarations and Initialization (DCL)

DCL00-C. const-qualify immutable objects

DCL01-C. Do not reuse variable names in subscopes

DCL02-C. Use visually distinct identifiers

DCL03-C. Use a static assertion to test the value of a constant expression

DCL04-C. Do not declare more than one variable per declaration

DCL05-C. Use type definitions to improve code readability

DCL06-C. Use meaningful symbolic constants to represent literal values in program logic

DCL07-C. Include the appropriate type information in function declarators

DCL08-C. Properly encode relationships in constant definitions

DCL09-C. Declare functions that return an errno error code with a return type of errno_t

DCL10-C. Maintain the contract between the writer and caller of variadic functions

DCL11-C. Understand the type issues associated with variadic functions

DCL12-C. Implement abstract data types using opaque types

DCL13-C. Declare function parameters that are pointers to values not changed by the function as const

DCL14-C. Do not make assumptions about the order of global variable initialization across translation units

DCL15-C. Declare objects that do not need external linkage with the storage-class specifier static

DCL30-C. Declare objects with appropriate storage durations

Table of Contents

DCL31-C. Declare identifiers before using them
DCL32-C. Guarantee that mutually visible identifiers are unique
DCL33-C. Ensure that restrict-qualified source and destination pointers in function arguments do not reference overlapping objects
DCL34-C. Use volatile for data that cannot be cached
DCL35-C. Do not convert a function using a type that does not match the function definition
DCL36-C. Do not declare an identifier with conficting linkage classifications
Chapter 4 Expressions (EXP)
EXP00-C. Use parentheses for precedence of operation
EXP01-C. Do not take the size of a pointer to determine the size of the pointed-to type
EXP02-C. Be aware of the short-circuit behavior of the logical AND and OR operators
EXP03-C. Do not assume the size of a structure is the sum of the sizes of its members
EXP04-C. Do not perform byte-by-byte comparisons between structures
EXP05-C. Do not cast away a const qualification
EXP06-C. Operands to the sizeof operator should not contain side effects
EXP07-C. Do not diminish the benefits of constants by assuming their values in expressions
EXP08-C. Ensure pointer arithmetic is used correctly
EXP09-C. Use sizeof to determine the size of a type or variable
EXP10-C. Do not depend on the order of evaluation of subexpressions or the order in which side effects take place
EXP11-C. Do not apply operators expecting one type to data of an

Table of Contents

incompatible type
EXP12-C. Do not ignore values returned by functions
EXP30-C. Do not depend on order of evaluation between sequence points
EXP31-C. Avoid side effects in assertions
EXP32-C. Do not cast away a volatile qualification
EXP33-C. Do not reference uninitialized memory
EXP34-C. Ensure a null pointer is not dereferenced
EXP35-C. Do not access or modify the result of a function call after a subsequent sequence point
EXP36-C. Do not convert pointers into more strictly aligned pointer types
EXP37-C. Call functions with the arguments intended by the API
EXP38-C. Do not call offsetof() on bit-field members or invalid types
Chapter 5 Integers (INT)
INT00-C. Understand the data model used by your implementation(s)
INT01-C. Use rsize_t or size_t for all integer values representing the size of an object
INT02-C. Understand integer conversion rules
INT03-C. Use a secure integer library
INT04-C. Enforce limits on integer values originating from untrusted sources
INT05-C. Do not use input functions to convert character data if they cannot handle all possible inputs
INT06-C. Use strtol() or a related function to convert a string token to an integer
INT07-C. Use only explicitly signed or unsigned char type for numeric

Table of Contents

values

INT08-C. Verify that all integer values are in range

INT09-C. Ensure enumeration constants map to unique values

INT10-C. Do not assume a positive remainder when using the % operator

INT11-C. Take care when converting from pointer to integer or integer to pointer

INT12-C. Do not make assumptions about the type of a plain int bit-field when used in an expression

INT13-C. Use bitwise operators only on unsigned operands

INT14-C. Avoid performing bitwise and arithmetic operations on the same data

INT15-C. Use intmax_t or uintmax_t for formatted I/O on programmer-defined integer types

INT30-C. Ensure that unsigned integer operations do not wrap

INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data

INT32-C. Ensure that operations on signed integers do not result in overflow

INT33-C. Ensure that division and modulo operations do not result in divide-by-zero errors

INT34-C. Do not shift a negative number of bits or more bits than exist in the operand

INT35-C. Evaluate integer expressions in a larger size before comparing or assigning to that size

Chapter 6 Floating Point (FLP)

FLP00-C. Understand the limitations of floating-point numbers

FLP01-C. Take care in rearranging floating-point expressions

Table of Contents

FLP02-C. Consider avoiding floating-point numbers when precise computation is needed
FLP03-C. Detect and handle floating-point errors
FLP30-C. Do not use floating-point variables as loop counters
FLP31-C. Do not call functions expecting real values with complex values
FLP32-C. Prevent or detect domain and range errors in math functions
FLP33-C. Convert integers to floating point for floating-point operations
FLP34-C. Ensure that floating-point conversions are within range of the new type
Chapter 7 Arrays (ARR)
ARR00-C. Understand how arrays work
ARR01-C. Do not apply the sizeof operator to a pointer when taking the size of an array
ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer
ARR30-C. Guarantee that array indices are within the valid range
ARR31-C. Use consistent array notation across all source files
ARR32-C. Ensure size arguments for variable length arrays are in a valid range
ARR33-C. Guarantee that copies are made into storage of sufficient size
ARR34-C. Ensure that array types in expressions are compatible
ARR35-C. Do not allow loops to iterate beyond the end of an array
ARR36-C. Do not subtract or compare two pointers that do not refer to the same array
ARR37-C. Do not add or subtract an integer to a pointer to a non-array object

Table of Contents

ARR38-C. Do not add or subtract an integer to a pointer if the resulting value does not refer to a valid array element

Chapter 8 Characters and Strings (STR)

STR00-C. Represent characters using an appropriate type

STR01-C. Adopt and implement a consistent plan for managing strings

STR02-C. Sanitize data passed to complex subsystems

STR03-C. Do not inadvertently truncate a null-terminated byte string

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

STR07-C. Use TR 24731 for remediation of existing string manipulation code

STR08-C. Use managed strings for development of new string manipulation code

STR30-C. Do not attempt to modify string literals

STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator

STR32-C. Null-terminate byte strings as required

STR33-C. Size wide character strings correctly

STR34-C. Cast characters to unsigned types before converting to larger integer sizes

STR35-C. Do not copy data from an unbounded source to a fixed-length array

STR36-C. Do not specify the bound of a character array initialized with a string literal

STR37-C. Arguments to character-handling functions must be

Table of Contents

representable as an unsigned char
Chapter 9 Memory Management (MEM)
MEM00-C. Allocate and free memory in the same module at the same level of abstraction
MEM01-C. Store a new value in pointers immediately after free()
MEM02-C. Immediately cast the result of a memory allocation function call into a pointer to the allocated type
MEM03-C. Clear sensitive information stored in reusable resources returned for reuse
MEM04-C. Do not perform zero-length allocations
MEM05-C. Avoid large stack allocations
MEM06-C. Ensure that sensitive data is not written out to disk
MEM07-C. Ensure that the arguments to calloc(), when multiplied, can be represented as a size_t
MEM08-C. Use realloc() only to resize dynamically allocated arrays
MEM09-C. Do not assume memory allocation routines initialize memory
MEM10-C. Use a pointer validation function
MEM30-C. Do not access freed memory
MEM31-C. Free dynamically allocated memory exactly once
MEM32-C. Detect and handle memory allocation errors
MEM33-C. Use the correct syntax for flexible array members
MEM34-C. Only free memory allocated dynamically
MEM35-C. Allocate sufficient memory for an object
Chapter 10 Input/Output (FIO)
FIO00-C. Take care when creating format strings
FIO01-C. Be careful using functions that use file names for identification

Table of Contents

FIO02-C. Canonicalize path names originating from untrusted sources

FIO03-C. Do not make assumptions about fopen() and file creation

FIO04-C. Detect and handle input and output errors

FIO05-C. Identify files using multiple file attributes

FIO06-C. Create files with appropriate access permissions

FIO07-C. Prefer fseek() to rewind()

FIO08-C. Take care when calling remove() on an open file

FIO09-C. Be careful with binary data when transferring data across systems

FIO10-C. Take care when using the rename() function

FIO11-C. Take care when specifying the mode parameter of fopen()

FIO12-C. Prefer setvbuf() to setbuf()

FIO13-C. Never push back anything other than one read character

FIO14-C. Understand the difference between text mode and binary mode with file streams

FIO15-C. Ensure that file operations are performed in a secure directory

FIO16-C. Limit access to files by creating a jail

FIO30-C. Exclude user input from format strings

FIO31-C. Do not simultaneously open the same file multiple times

FIO32-C. Do not perform operations on devices that are only appropriate for files

FIO33-C. Detect and handle input output errors resulting in undefined behavior

FIO34-C. Use int to capture the return value of character I/O functions

FIO35-C. Use feof() and ferror() to detect end-of-file and file errors

Table of Contents

when sizeof(int) == sizeof(char)

FIO36-C. Do not assume a new-line character is read when using fgets()

FIO37-C. Do not assume character data has been read

FIO38-C. Do not use a copy of a FILE object for input and output

FIO39-C. Do not alternately input and output from a stream without an intervening flush or positioning call

FIO40-C. Reset strings on fgets() failure

FIO41-C. Do not call getc() or putc() with stream arguments that have side effects

FIO42-C. Ensure files are properly closed when they are no longer needed

FIO43-C. Do not create temporary files in shared directories

FIO44-C. Only use values for fsetpos() that are returned from fgetpos()

Chapter 11 Environment (ENV)

ENV00-C. Do not store the pointer to the string returned by getenv()

ENV01-C. Do not make assumptions about the size of an environment variable

ENV02-C. Beware of multiple environment variables with the same effective name

ENV03-C. Sanitize the environment when invoking external programs

ENV04-C. Do not call system() if you do not need a command processor

ENV30-C. Do not modify the string returned by getenv()

ENV31-C. Do not rely on an environment pointer following an operation that may invalidate it

ENV32-C. No atexit handler should terminate in any way other than by

Table of Contents

returning

Chapter 12 Signals (SIG)

SIG00-C. Mask signals handled by noninterruptible signal handlers

SIG01-C. Understand implementation-specific details regarding signal handler persistence

SIG02-C. Avoid using signals to implement normal functionality

SIG30-C. Call only asynchronous-safe functions within signal handlers

SIG31-C. Do not access or modify shared objects in signal handlers

SIG32-C. Do not call longjmp() from inside a signal handler

SIG33-C. Do not recursively invoke the raise() function

SIG34-C. Do not call signal() from within interruptible signal handlers

Chapter 13 Error Handling (ERR)

ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy

ERR01-C. Use ferror() rather than errno to check for FILE stream errors

ERR02-C. Avoid in-band error indicators

ERR03-C. Use runtime-constraint handlers when calling functions defined by TR 24731-1

ERR04-C. Choose an appropriate termination strategy

ERR05-C. Application-independent code should provide error detection without dictating error handling

ERR06-C. Understand the termination behavior of assert() and abort()

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

ERR31-C. Do not redefine errno

ERR32-C. Do not rely on indeterminate values of errno

Chapter 14 Miscellaneous (MSC)

Table of Contents

MSC00-C. Compile cleanly at high warning levels
MSC01-C. Strive for logical completeness
MSC02-C. Avoid errors of omission
MSC03-C. Avoid errors of addition
MSC04-C. Use comments consistently and in a readable fashion
MSC05-C. Do not manipulate time_t typed values directly
MSC06-C. Be aware of compiler optimization when dealing with sensitive data
MSC07-C. Detect and remove dead code
MSC08-C. Library functions should validate their parameters
MSC09-C. Character encoding: use subset of ASCII for safety
MSC10-C. Character encoding: UTF-8-related issues
MSC11-C. Incorporate diagnostic tests using assertions
MSC12-C. Detect and remove code that has no effect
MSC13-C. Detect and remove unused values
MSC14-C. Do not introduce unnecessary platform dependencies
MSC15-C. Do not depend on undefined behavior
MSC30-C. Do not use the rand() function for generating pseudorandom numbers
MSC31-C. Ensure that return values are compared against the proper type

Appendix: POSIX (POS)

POS00-C. Avoid race conditions with multiple threads
POS01-C. Check for the existence of links
POS02-C. Follow the principle of least privilege
POS30-C. Use the readlink() function properly
POS31-C. Do not unlock or destroy another thread's mutex

Table of Contents

POS32-C. Include a mutex when using bit-fields in a multithreaded environment

POS33-C. Do not use vfork()

POS34-C. Do not call putenv() with a pointer to an automatic variable as the argument

POS35-C. Avoid race conditions while checking for the existence of a symbolic link

POS36-C. Observe correct revocation order while relinquishing privileges

POS37-C. Ensure that privilege relinquishment is successful

Glossary

References

Index