POS54-C. Detect and handle POSIX library errors

All standard library functions, including I/O functions and memory allocation functions, return either a valid value or a value of the correct return type that indicates an error (for example, 1 or a null pointer). Assuming that all calls to such functions will succeed and failing to check the return value for an indication of an error is a dangerous practice that may lead to unexpected or undefined behavior when an error occurs. It is essential that programs detect and appropriately handle all errors in accordance with an error-handling policy, as discussed in ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy. In addition to the C standard library functions mentioned in ERR33-C. Detect and handle standard library errors, the following functions defined in POSIX require error checking (list is not all-inclusive).

The successful completion or failure of each of the standard library functions listed in the following table shall be determined either by comparing the function's return value with the value listed in the column labeled "Error Return" or by calling one of the library functions mentioned in the footnotes to the same column.

Function	Successful Return	Error Return	`errno`
`fmemopen()`	Pointer to a `FILE` object	`NULL`	`ENOMEM`
`open_memstream()`	Pointer to a `FILE` object	`NULL`	`ENOMEM`
`posix_memalign()`	`0`	Nonzero	Unchanged

Setting `errno` is a POSIX [ISO/IEC 9945:2008] extension to the C Standard. On error, `posix_memalign()` returns a value that corresponds to one of the constants defined in the `<errno.h>` header. The function does not set `errno`. The `posix_memalign()` function is optional and is not required to be provided by POSIX-conforming implementations.

Noncompliant Code Example (POSIX)

In this noncompliant code example, `fmemopen()` and `open_memstream()` are assumed to succeed. However, if the calls fail, the two file pointers `in` and `out` will be null and the program will have undefined behavior.

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
  FILE *out;
  FILE *in;
  size_t size;
  char *ptr;

  if (argc != 2) {
    /* Handle error */
  }

  in = fmemopen(argv[1], strlen(argv[1]), "r");
  /* Use in */

  out = open_memstream(&ptr, &size);
  /* Use out */

  return 0;
}
```

Compliant Solution (POSIX)

A compliant solution avoids assuming that `fmemopen()` and `open_memstream()` succeed regardless of its arguments and tests the return value of the function before using the file pointers `in` and `out`:

```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[]) {
  FILE *out;
  FILE *in;
  size_t size;
  char *ptr;

  if (argc != 2) {
    /* Handle error */
  }

  in = fmemopen(argv[1], strlen(argv[1]), "r");

  if (in == NULL){
    /* Handle error */
  }
  /* Use in */

  out = open_memstream(&ptr, &size);

  if (out == NULL){
    /* Handle error */
  }
  /* Use out */
  return 0;
}
```

Exceptions

ERR33-C-EX1: The exception from EXP12-C. Do not ignore values returned by functions still applies. If the return value is inconsequential or if any errors can be safely ignored, such as for functions called because of their side effects, the function should be explicitly cast to `void` to signify programmer intent.

ERR33-C-EX2: Ignore the return value of a function that cannot fail or whose return value cannot signify that an error condition need not be diagnosed. For example, `strcpy()` is one such function.

Return values from the following functions do not need to be checked because their historical use has overwhelmingly omitted error checking, and the consequences are not relevant to security.

Function	Successful Return	Error Return
`printf()`	Number of characters (nonnegative)	Negative
`putchar()`	Character written	`EOF`
`puts()`	Nonnegative	`EOF` (negative)
`putwchar()`	Wide character written	`WEOF`
`vprintf()`	Number of characters (nonnegative)	Negative
`vwprintf()`	Number of wide characters (nonnegative)	Negative
`wprintf()`	Number of wide characters (nonnegative)	Negative

Risk Assessment

Failing to detect error conditions can lead to unpredictable results, including abnormal program termination and denial-of-service attacks or, in some situations, could even allow an attacker to run arbitrary code.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
POS54-C	High	Likely	Medium	P18	L1

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-POS54**	
CodeSonar	5.1p0	**LANG.FUNCS.IRV**	Ignored return value
Compass /ROSE			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
Coverity	2017.07	**CHECKED_RETURN**	Finds inconsistencies in how function call return values are handled. Coverity Prevent cannot discover all violations of this recommendation, so further verification is necessary
Klocwork	**2018**	**SV.RVT. RETVAL_NOTTES TED**	
LDRA tool suite	9.7.1	**80 D**	Partially implemented
Parasoft C /C++test	10.4.2	**CERT_C-POS54-a** **CERT_C-POS54-b** **CERT_C-POS54-c**	The value returned by a function having non-void return type shall be used The value returned by a function having non-void return type shall be used Avoid null pointer dereferencing
Polyspace Bug Finder	R2019b	CERT C: Rule POS54-C	Checks for situations where return value of a sensitive function is not checked (rule fully covered)
PRQA QA-C	9.5	**3200**	Partially implemented

Related Vulnerabilities

The vulnerability in Adobe Flash [VU#159523] arises because Flash neglects to check the return value from `calloc()`. Even when `calloc()` returns `NUL`, Flash writes to an offset from the return value. Dereferencing `NULL` usually results in a program crash, but dereferencing an offset from `NULL` allows an exploit to succeed without crashing the program.

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

Key here (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
CERT C Secure Coding Standard	API04-C. Provide a consistent and usable error-checking mechanism ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy ERR02-C. Avoid in-band error indicators ERR05-C. Application-independent code should provide error detection without dictating error handling EXP12-C. Do not ignore values returned by functions EXP34-C. Do not dereference null pointers FIO10-C. Take care when using the rename() function FIO13-C. Never push back anything other than one read character FIO33-C. Detect and handle input output errors resulting in undefined behavior FIO34-C. Distinguish between characters read from a file and EOF or WEOF FLP03-C. Detect and handle floating-point errors FLP32-C. Prevent or detect domain and range errors in math functions MEM04-C. Do not perform zero-length allocations MEM12-C. Consider using a goto chain when leaving a function on error when using and releasing resources	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT C	ERR10-CPP. Check for error conditions FIO04-CPP. Detect and handle input and output errors	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TS 17961	Failing to detect and handle standard library errors [liberr]	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-252, Unchecked return value	2017-07-06: CERT: Partial overlap
CWE 2.11	CWE-253, Incorrect check of function return value	2017-07-06: CERT: Partial overlap
CWE 2.11	CWE-391, Unchecked error condition	2017-07-06: CERT: Rule subset of CWE

Bibliography

[DHS 2006]	Handle All Errors Safely
[Henricson 1997]	Recommendation 12.1, "Check for All Errors Reported from Functions"
[ISO/IEC 9899:2011]	Subclause 7.21.7.10, "The `ungetc` Function"