ERR34-C. Detect errors when converting a string to a number

The process of parsing an integer or floating-point number from a string can produce many errors. The string might not contain a number. It might contain a number of the correct type that is out of range (such as an integer that is larger than `INT_MAX`). The string may also contain extra information after the number, which may or may not be useful after the conversion. These error conditions must be detected and addressed when a string-to-number conversion is performed using a C Standard Library function.

The `strtol()`, `strtoll()`, `strtoimax()`, `strtoul()`, `strtoull()`, `strtoumax()`, `strtof()`, `strtod()`, and `strtold()` functions convert the initial portion of a null-terminated byte string to a `long int`, `long long int`, `intmax_t`, `unsigned long int`, `unsigned long long int`, `uintmax_t`, `float`, `double`, and `long double` representation, respectively.

Use one of the C Standard Library `strto*()` functions to parse an integer or floating-point number from a string. These functions provide more robust error handling than alternative solutions. Also, use the `strtol()` function to convert to a smaller signed integer type such as `signed int`, `signed short`, and `signed char`, testing the result against the range limits for that type. Likewise, use the `strtoul()` function to convert to a smaller unsigned integer type such as `unsigned int`, `unsigned short`, and `unsigned char`, and test the result against the range limits for that type. These range tests do nothing if the smaller type happens to have the same size and representation for a particular implementation.

Noncompliant Code Example (`atoi()`)

This noncompliant code example converts the string token stored in the `buff` to a signed integer value using the `atoi()` function:

```
#include <stdlib.h>

void func(const char *buff) {
  int si;

  if (buff) {
    si = atoi(buff);
  } else {
    /* Handle error */
  }
}
```

The `atoi()`, `atol()`, `atoll()`, and `atof()` functions convert the initial portion of a string token to `int`, `long int`, `long long int`, and `double` rep resentation, respectively. Except for the behavior on error, they are equivalent to

```
atoi: (int)strtol(nptr, (char **)NULL, 10)
atol: strtol(nptr, (char **)NULL, 10)
atoll: strtoll(nptr, (char **)NULL, 10)
atof: strtod(nptr, (char **)NULL)
```

Unfortunately, `atoi()` and related functions lack a mechanism for reporting errors for invalid values. Specifically, these functions:

- do not need to set `errno` on an error;
- have undefined behavior if the value of the result cannot be represented;
- return 0 (or 0.0) if the string does not represent an integer (or decimal), which is indistinguishable from a correctly formatted, zero-denoting input string.

Noncompliant Example (`sscanf()`)

This noncompliant example uses the `sscanf()` function to convert a string token to an integer. The `sscanf()` function has the same limitations as `atoi(`
:

```
#include <stdio.h>

void func(const char *buff) {
  int matches;
  int si;

  if (buff) {
    matches = sscanf(buff, "%d", &si);
    if (matches != 1) {
      /* Handle error */
    }
  } else {
    /* Handle error */
  }
}
```

The `sscanf()` function returns the number of input items successfully matched and assigned, which can be fewer than provided for, or even 0 in the event of an early matching failure. However, `sscanf()` fails to report the other errors reported by `strtol()`, such as numeric overflow.

Compliant Solution (`strtol()`)

The `strtol()`, `strtoll()`, `strtoimax())`, `strtoul()`, `strtoull()`, `strtoumax()`, `strtof()`, `strtod()`, and `strtold()` functions convert a null-terminated byte string to `long int`, `long long int`, `intmax_t`, `unsigned long int`, `unsigned long long int`, `uintmax_t`, `float`, `double`, and `long double` representation, respectively.

This compliant solution uses `strtol()` to convert a string token to an integer and ensures that the value is in the range of `int`:

```
#include <errno.h>
#include <limits.h>
#include <stdlib.h>
#include <stdio.h>

void func(const char *buff) {
  char *end;
  int si;

  errno = 0;

  const long sl = strtol(buff, &end, 10);

  if (end == buff) {
    fprintf(stderr, "%s: not a decimal number\n", buff);
  } else if ('\0' != *end) {
    fprintf(stderr, "%s: extra characters at end of input: %s\n", buff, end);
  } else if ((LONG_MIN == sl || LONG_MAX == sl) && ERANGE == errno) {
    fprintf(stderr, "%s out of range of type long\n", buff);
  } else if (sl > INT_MAX) {
    fprintf(stderr, "%ld greater than INT_MAX\n", sl);
  } else if (sl < INT_MIN) {
     fprintf(stderr, "%ld less than INT_MIN\n", sl);
  } else {
    si = (int)sl;

    /* Process si */
  }
}
```

Risk Assessment

It is rare for a violation of this rule to result in a security vulnerability unless it occurs in security-sensitive code. However, violations of this rule can easily result in lost or misinterpreted data.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
ERR34-C	Medium	Unlikely	Medium	**P4**	**L3**

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-ERR34**	
Clang	3.9	`cert-err34-c`	Checked by `clang-tidy`
CodeSonar	5.2p0	**BADFUNC.ATOF** **BADFUNC.ATOI** **BADFUNC.ATOL** **BADFUNC.ATOLL** **(customization)**	Use of atof Use of atoi Use of atol Use of atoll Users can add custom checks for uses of other undesirable conversion functions.
Compass/ROSE			Can detect violations of this recommendation by flagging invocations of the following functions: • `atoi()` • `scanf()`, `fscanf()`, `sscanf()` • Others?
Klocwork	2018	**MISRA.STDLIB.ATOI**	
LDRA tool suite	9.7.1	**44 S**	Fully implemented
Parasoft C/C++test	10.4.2	**CERT_C-ERR34-a**	The library functions atof, atoi and atol from library stdlib.h shall not be used
Polyspace Bug Finder	R2019b	CERT C: Rule ERR34-C	Checks for unsafe conversion from string to numeric value (rule fully covered)
PRQA QA-C	9.7	**5030**	Partially implemented
PRQA QA-C++	4.4	**5016**	
SonarQube C/C++ Plugin	3.11	**S989**	

Related Vulnerabilities

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	INT06-CPP. Use strtol() or a related function to convert a string token to an integer	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-676, Use of potentially dangerous function	2017-05-18: CERT: Rule subset of CWE
CWE 2.11	CWE-758	2017-06-29: CERT: Partial overlap

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-20 and ERR34-C

Intersection(ERR34-C, CWE-20) = Ø

CERT C does not define the concept of ‘input validation’. String-to-integer conversion (ERR34-C) may qualify as input validation, but this is outside the scope of the CERT rule.

CWE-391 and ERR34-C

CWE-391 = Union(ERR34-C, list) where list =

- Failure to errors outside of string-to-number conversion functions

CWE-676 and ERR34-C

- Independent(ENV33-C, CON33-C, STR31-C, EXP33-C, MSC30-C, ERR34-C)

- ERR34-C implies that string-parsing functions (eg atoi() and scanf()) are dangerous.

- CWE-676 = Union(ERR34-C, list) where list =

- Invocation of dangerous functions besides the following:

- atoi(), atol(), atoll(), atof(), The scanf()family

CWE-758 and ERR34-C

Independent(INT34-C, INT36-C, MSC37-C, FLP32-C, EXP33-C, EXP30-C, ERR34-C, ARR32-C)

Intersection(CWE-758, ERR34-C) =

- Undefined behavior arising from a non-representable numeric value being parsed by an ato*() or scanf() function

CWE-758 – ERR34-C =

- Undefined behavior arising from using a function outside of the ato*() or scanf() family

ERR34-C – CWE-758 =

- The ato*() or scanf() family receives input that is not a number when trying to parse one

Bibliography

[ISO/IEC 9899:2011]	Subclause 7.22.1, "Numeric conversion functions" Subclause 7.21.6, "Formatted input/output functions"
[Klein 2002]