STR37-C. Arguments to character-handling functions must be representable as an unsigned char

According to the C Standard, 7.4 [ISO/IEC 9899:2011],

> *The header `<ctype.h>` declares several functions useful for classifying and mapping characters. In all cases the argument is an `int`, the value of which shall be representable as an `unsigned char` or shall equal the value of the macro EOF. If the argument has any other value, the behavior is undefined.*

See also undefined behavior 113.

This rule is applicable only to code that runs on platforms where the `char` data type is defined to have the same range, representation, and behavior as `signed char`.

Following are the character classification functions that this rule addresses:

`isalnum()`	`isalpha()`	`isascii()`XSI	`isblank()`
`iscntrl()`	`isdigit()`	`isgraph()`	`islower()`
`isprint()`	`ispunct()`	`isspace()`	`isupper()`
`isxdigit()`	`toascii()`XSI	`toupper()`	`tolower()`

XSI denotes an X/Open System Interfaces Extension to ISO/IEC 9945—POSIX. These functions are not defined by the C Standard.

This rule is a specific instance of STR34-C. Cast characters to unsigned char before converting to larger integer sizes.

Noncompliant Code Example

On implementations where plain `char` is signed, this code example is noncompliant because the parameter to `isspace()`, `*t`, is defined as a `const char *`, and this value might not be representable as an `unsigned char`:

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

size_t count_preceding_whitespace(const char *s) {
  const char *t = s;
  size_t length = strlen(s) + 1;
  while (isspace(*t) && (t - s < length)) {
    ++t;
  }
  return t - s;
}
```

The argument to `isspace()` must be `EOF` or representable as an `unsigned char`; otherwise, the result is undefined.

Compliant Solution

This compliant solution casts the character to `unsigned char` before passing it as an argument to the `isspace()` function:

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

size_t count_preceding_whitespace(const char *s) {
  const char *t = s;
  size_t length = strlen(s) + 1;
  while (isspace((unsigned char)*t) && (t - s < length)) {
    ++t;
  }
  return t - s;
}
```

Risk Assessment

Passing values to character handling functions that cannot be represented as an `unsigned char` to character handling functions is undefined behavior.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
STR37-C	Low	Unlikely	Low	**P3**	**L3**

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**ctype-limits**	Partially checked
Axivion Bauhaus Suite	6.9.0	**CertC-STR37**	Fully implemented
CodeSonar	5.1p0	**MISC.NEGCHAR**	Negative character value
Compass/ROSE			Could detect violations of this rule by seeing if the argument to a character handling function (listed above) is not an `unsigned char`
ECLAIR	1.2	**CC2.STR37**	Fully implemented
LDRA tool suite	9.7.1	**663 S**	Fully implemented
Parasoft C /C++test	10.4.2	**CERT_C-STR37-a**	Do not pass incorrect values to ctype.h library functions
Polyspace Bug Finder	R2019b	CERT C: Rule STR37-C	Checks for invalid use of standard library integer routine (rule fully covered)
PRQA QA-C	9.7	**4413, 4414**	Fully implemented
PRQA QA-C++	4.4	**3051**	
RuleChecker	19.04	**ctype-limits**	Partially checked
TrustInSoft Analyzer	1.38	**valid_char**	Partially verified.

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 Secure Coding Standard	STR34-C. Cast characters to unsigned char before converting to larger integer sizes	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TS 17961	Passing arguments to character-handling functions that are not representable as unsigned char [chrsgnext]	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-704, Incorrect Type Conversion or Cast	2017-06-14: CERT: Rule subset of CWE

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-686 and STR37-C

Intersection(CWE-686, STR37-C) = Ø

STR37-C is not about the type of the argument passed (which is signed int), but about the restrictions placed on the value in this type (must be 0-UCHAR_MAX or EOF). I interpret ‘argument type’ to be specific to the C language, so CWE-686 does not apply to incorrect argument values, just incorrect types (which is relatively rare in C, but still possible).

CWE-704 and STR37-C

STR37-C = Subset(STR34-C)

CWE-683 and STR37-C

Intersection(CWE-683, STR37-C) = Ø

STR37-C excludes mis-ordered function arguments (assuming they pass type-checking), because there is no easy way to reliably detect violations of CWE-683.

Bibliography

[ISO/IEC 9899:2011]	7.4, "Character Handling `<ctype.h>`"
[Kettlewell 2002]	Section 1.1, "`<ctype.h>` and Characters Types"