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

Signed character data must be converted to `unsigned char` before being assigned or converted to a larger signed type. This rule applies to both `signed char` and (plain) `char` characters on implementations where `char` is defined to have the same range, representation, and behaviors as `signed char`.

However, this rule is applicable only in cases where the character data may contain values that can be interpreted as negative numbers. For example, if the `char` type is represented by a two's complement 8-bit value, any character value greater than +127 is interpreted as a negative value.

This rule is a generalization of STR37-C. Arguments to character-handling functions must be representable as an unsigned char.

Noncompliant Code Example

This noncompliant code example is taken from a vulnerability in bash versions 1.14.6 and earlier that led to the release of CERT Advisory CA-1996-22. This vulnerability resulted from the sign extension of character data referenced by the `c_str` pointer in the `yy_string_get()` function in the `parse.y` module of the bash source code:

```
static int yy_string_get(void) {
  register char *c_str;
  register int c;

  c_str = bash_input.location.string;
  c = EOF;

  /* If the string doesn't exist or is empty, EOF found */
  if (c_str && *c_str) {
    c = *c_str++;
    bash_input.location.string = c_str;
  }
  return (c);
}
```

The `c_str` variable is used to traverse the character string containing the command line to be parsed. As characters are retrieved from this pointer, they are stored in a variable of type `int`. For implementations in which the `char` type is defined to have the same range, representation, and behavior as `signed char`, this value is sign-extended when assigned to the `int` variable. For character code 255 decimal (1 in two's complement form), this sign extension results in the value 1 being assigned to the integer, which is indistinguishable from `EOF`.

Noncompliant Code Example

This problem can be repaired by explicitly declaring the `c_str` variable as `unsigned char`:

```
static int yy_string_get(void) {
  register unsigned char *c_str;
  register int c;

  c_str = bash_input.location.string;
  c = EOF;

  /* If the string doesn't exist or is empty, EOF found */
  if (c_str && *c_str) {
    c = *c_str++;
    bash_input.location.string = c_str;
  }
  return (c);
}
```

This example, however, violates STR04-C. Use plain char for characters in the basic character set.

Compliant Solution

In this compliant solution, the result of the expression `*c_str++` is cast to `unsigned char` before assignment to the `int` variable `c`:

```
static int yy_string_get(void) {
  register char *c_str;
  register int c;

  c_str = bash_input.location.string;
  c = EOF;

  /* If the string doesn't exist or is empty, EOF found */
  if (c_str && *c_str) {
    /* Cast to unsigned type */
    c = (unsigned char)*c_str++;

    bash_input.location.string = c_str;
  }
  return (c);
}
```

Noncompliant Code Example

In this noncompliant code example, the cast of `*s` to `unsigned int` can result in a value in excess of `UCHAR_MAX` because of integer promotions, a violation of ARR30-C. Do not form or use out-of-bounds pointers or array subscripts:

```
#include <limits.h>
#include <stddef.h>

static const char table[UCHAR_MAX + 1] = { 'a' /* ... */ };

ptrdiff_t first_not_in_table(const char *c_str) {
  for (const char *s = c_str; *s; ++s) {
    if (table[(unsigned int)*s] != *s) {
      return s - c_str;
    }
  }
  return -1;
}
```

Compliant Solution

This compliant solution casts the value of type `char` to `unsigned char` before the implicit promotion to a larger type:

```
#include <limits.h>
#include <stddef.h>

static const char table[UCHAR_MAX + 1] = { 'a' /* ... */ };

ptrdiff_t first_not_in_table(const char *c_str) {
  for (const char *s = c_str; *s; ++s) {
    if (table[(unsigned char)*s] != *s) {
      return s - c_str;
    }
  }
  return -1;
}
```

Risk Assessment

Conversion of character data resulting in a value in excess of `UCHAR_MAX` is an often-missed error that can result in a disturbingly broad range of potentially severe vulnerabilities.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
STR34-C	Medium	Probable	Medium	P8	L2

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**char-sign-conversion**	Fully checked
Axivion Bauhaus Suite	6.9.0	**CertC-STR34**	Fully implemented
CodeSonar	5.2p0	**MISC.NEGCHAR**	Negative Character Value
Compass/ROSE			Can detect violations of this rule when checking for violations of INT07-C. Use only explicitly signed or unsigned char type for numeric values
Coverity	2017.07	**MISRA C 2012 Rule 10.1** **MISRA C 2012 Rule 10.2** **MISRA C 2012 Rule 10.3** **MISRA C 2012 Rule 10.4**	Implemented Essential type checkers
ECLAIR	1.2	**CC2.STR34**	Fully implemented
GCC	2.95 and later	`-Wchar-subscripts`	Detects objects of type `char` used as array indices
LDRA tool suite	9.7.1	**434 S**	Partially implemented
Parasoft C/C++test	10.4.2	**CERT_C-STR34-a** **CERT_C-STR34-b** **CERT_C-STR34-c** **CERT_C-STR34-d** **CERT_C-STR34-e** **CERT_C-STR34-f**	There shall be no implicit conversions from integral to floating type Cast characters to unsigned char before assignment to larger integer sizes An expressions of the 'signed char' type should not be used as an array index Cast characters to unsigned char before converting to larger integer sizes Avoid implicit conversions from floating to integral type A cast should not be performed between a pointer to object type and a different pointer to object type
Polyspace Bug Finder	R2019b	CERT C: Rule STR34-C	Checks for misuse of sign-extended character value (rule fully covered)
PRQA QA-C	9.7	**2140, 2141, 2143, 2144,** **2145, 2147, 2148, 2149,** **2151, 2152, 2153, 2155**	Fully implemented
PRQA QA-C++	4.4	**3051**	
RuleChecker	19.04	**char-sign-conversion**	Fully checked
TrustInSoft Analyzer	1.38	**out of bounds read**	Partially verified (exhaustively detects undefined behavior).

Related Vulnerabilities

CVE-2009-0887 results from a violation of this rule. In Linux PAM (up to version 1.0.3), the `libpam` implementation of `strtok()` casts a (potentially signed) character to an integer for use as an index to an array. An attacker can exploit this vulnerability by inputting a string with non-ASCII characters, causing the cast to result in a negative index and accessing memory outside of the array [xorl 2009].

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

Related Guidelines

CERT C Secure Coding Standard	STR37-C. Arguments to character-handling functions must be representable as an unsigned char STR04-C. Use plain char for characters in the basic character set ARR30-C. Do not form or use out-of-bounds pointers or array subscripts
ISO/IEC TS 17961:2013	Conversion of signed characters to wider integer types before a check for EOF [signconv]

MISRA-C:2012	Rule 10.1 (required) Rule 10.2 (required) Rule 10.3 (required) Rule 10.4 (required)
MITRE CWE	CWE-704, Incorrect Type Conversion or Cast

Bibliography

[xorl 2009]	CVE-2009-0887: Linux-PAM Signedness Issue