# MSC33-C. Do not pass invalid data to the asctime() function

The C Standard, 7.27.3.1 [ISO/IEC 9899:2011], provides the following sample implementation of the `asctime()` function:

```
char *asctime(const struct tm *timeptr) {
  static const char wday_name[7][3] = {
    "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
  };
  static const char mon_name[12][3] = {
    "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
  };
  static char result[26];
  sprintf(
    result,
    "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
    wday_name[timeptr->tm_wday],
    mon_name[timeptr->tm_mon],
    timeptr->tm_mday, timeptr->tm_hour,
    timeptr->tm_min, timeptr->tm_sec,
    1900 + timeptr->tm_year
  );
  return result;
}
```

This function is supposed to output a character string of 26 characters at most, including the terminating null character. If we count the length indicated by the format directives, we arrive at 25. Taking into account the terminating null character, the array size of the string appears sufficient.

However, this implementation assumes that the values of the `struct tm` data are within normal ranges and does nothing to enforce the range limit. If any of the values print more characters than expected, the `sprintf()` function may overflow the `result` array. For example, if `tm_year` has the value `12345` then 27 characters (including the terminating null character) are printed, resulting in a buffer overflow.

The *POSIX® Base Specifications* [IEEE Std 1003.1:2013] says the following about the `asctime()` and `asctime_r()` functions:

> *These functions are included only for compatibility with older implementations. They have undefined behavior if the resulting string would be too long, so the use of these functions should be discouraged. On implementations that do not detect output string length overflow, it is possible to overflow the output buffers in such a way as to cause applications to fail, or possible system security violations. Also, these functions do not support localized date and time formats. To avoid these problems, applications should use `strftime()` to generate strings from broken-down times.*

The C Standard, Annex K, also defines `asctime_s()`, which can be used as a secure substitute for `asctime()`.

The `asctime()` function appears in the list of obsolescent functions in MSC24-C. Do not use deprecated or obsolescent functions.

## Noncompliant Code Example

This noncompliant code example invokes the `asctime()` function with potentially unsanitized data:

```
#include <time.h>

void func(struct tm *time_tm) {
  char *time = asctime(time_tm);
  /* ... */
}
```

## Compliant Solution (`strftime()`)

The `strftime()` function allows the programmer to specify a more rigorous format and also to specify the maximum size of the resulting time string:

```
#include <time.h>

enum { maxsize = 26 };

void func(struct tm *time) {
  char s[maxsize];
  /* Current time representation for locale */
  const char *format = "%c";

  size_t size = strftime(s, maxsize, format, time);
}
```

This call has the same effects as `asctime()` but also ensures that no more than `maxsize` characters are printed, preventing buffer overflow.

## Compliant Solution (`asctime_s()`)

The C Standard, Annex K, defines the `asctime_s()` function, which serves as a close replacement for the `asctime()` function but requires an additional argument that specifies the maximum size of the resulting time string:

```
#define __STDC_WANT_LIB_EXT1__ 1
#include <time.h>

enum { maxsize = 26 };

void func(struct tm *time_tm) {
  char buffer[maxsize];

  if (asctime_s(buffer, maxsize, &time_tm)) {
    /* Handle error */
  }
}
```

## Risk Assessment

On implementations that do not detect output-string-length overflow, it is possible to overflow the output buffers.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
|---|---|---|---|---|---|
| MSC33-C | High | Likely | Low | **P27** | **L1** |

### Automated Detection

| Tool | Version | Checker | Description |
|---|---|---|---|
| Astrée | 19.04 | | Supported, but no explicit checker |
| Axivion Bauhaus Suite | 6.9.0 | **CertC-MSC33** | |
| LDRA tool suite | 9.7.1 | **44 S** | Enhanced Enforcement |
| Parasoft C/C++test | 10.4.2 | **CERT_C-MSC33-a** | Avoid functions which use time from standard C library |
| Polyspace Bug Finder | R2019b | CERT C: Rule MSC33-C | Checks for use of obsolete standard function (rule fully covered) |
| PRQA QA-C | 9.7 | **5032** | |
| PRQA QA-C++ | 4.4 | **5030** | |
| RuleChecker | 19.04 | | Supported, but no explicit checker |

### 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 | MSC24-C. Do not use deprecated or obsolescent functions | Prior to 2018-01-12: CERT: Unspecified Relationship |

## Bibliography

| | |
|---|---|
| [IEEE Std 1003.1:2013] | XSH, System Interfaces, `asctime` |
| [ISO/IEC 9899:2011] | 7.27.3.1, "The `asctime` Function" |