

CON33-C. Avoid race conditions when using library functions

Some C standard library functions are not guaranteed to be [reentrant](#) with respect to threads. Functions such as `strtok()` and `asctime()` return a pointer to the result stored in function-allocated memory on a per-process basis. Other functions such as `rand()` store state information in function-allocated memory on a per-process basis. Multiple threads invoking the same function can cause concurrency problems, which often result in abnormal behavior and can cause more serious [vulnerabilities](#), such as [abnormal termination](#), [denial-of-service attack](#), and data integrity violations.

According to the C Standard, the library functions listed in the following table may contain data races when invoked by multiple threads.

Functions	Remediation
<code>rand()</code> , <code>srand()</code>	MSC30-C. Do not use the <code>rand()</code> function for generating pseudorandom numbers
<code>getenv()</code> , <code>getenv_s()</code>	ENV34-C. Do not store pointers returned by certain functions
<code>strtok()</code>	<code>strtok_s()</code> in C11 Annex K <code>strtok_r()</code> in POSIX
<code>strerror()</code>	<code>strerror_s()</code> in C11 Annex K <code>strerror_r()</code> in POSIX
<code>asctime()</code> , <code>ctime()</code> , <code>localtime()</code> , <code>gmtime()</code>	<code>asctime_s()</code> , <code>ctime_s()</code> , <code>localtime_s()</code> , <code>gmtime_s()</code> in C11 Annex K
<code>setlocale()</code>	Protect multithreaded access to locale-specific functions with a mutex
<code>ATOMIC_VAR_INIT</code> , <code>atomic_init()</code>	Do not attempt to initialize an atomic variable from multiple threads
<code>tmpnam()</code>	<code>tmpnam_s()</code> in C11 Annex K <code>tmpnam_r()</code> in POSIX
<code>mbrtoc16()</code> , <code>c16rtomb()</code> , <code>mbrtoc32()</code> , <code>c32rtomb()</code>	Do not call with a null <code>mbstate_t *</code> argument

Section 2.9.1 of the *Portable Operating System Interface (POSIX®), Base Specifications, Issue 7* [IEEE Std 1003.1:2013] extends the list of functions that are not required to be thread-safe.

Noncompliant Code Example

In this noncompliant code example, the function `f()` is called from within a multithreaded application but encounters an error while calling a system function. The `strerror()` function returns a human-readable error string given an error number. The C Standard, 7.24.6.2 [ISO/IEC 9899:2011], specifically states that `strerror()` is not required to avoid data races. An [implementation](#) could write the error string into a static array and return a pointer to it, and that array might be accessible and modifiable by other threads.

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

void f(FILE *fp) {
    fpos_t pos;
    errno = 0;

    if (0 != fgetpos(fp, &pos)) {
        char *errmsg = strerror(errno);
        printf("Could not get the file position: %s\n", errmsg);
    }
}
```

This code first sets `errno` to 0 to comply with [ERR30-C. Set `errno` to zero before calling a library function known to set `errno`, and check `errno` only after the function returns a value indicating failure.](#)

Compliant Solution (Annex K, `strerror_s()`)

This compliant solution uses the `strerror_s()` function from Annex K of the C Standard, which has the same functionality as `strerror()` but guarantees thread-safety:

```

#define __STDC_WANT_LIB_EXT1__ 1
#include <errno.h>
#include <stdio.h>
#include <string.h>

enum { BUFFERSIZE = 64 };
void f(FILE *fp) {
    fpos_t pos;
    errno = 0;

    if (0 != fgetpos(fp, &pos)) {
        char errmsg[BUFFERSIZE];
        if (strerror_s(errmsg, BUFFERSIZE, errno) != 0) {
            /* Handle error */
        }
        printf("Could not get the file position: %s\n", errmsg);
    }
}

```

Because Annex K is optional, `strerror_s()` may not be available in all implementations.

Compliant Solution (POSIX, `strerror_r()`)

This compliant solution uses the POSIX `strerror_r()` function, which has the same functionality as `strerror()` but guarantees thread safety:

```

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

enum { BUFFERSIZE = 64 };

void f(FILE *fp) {
    fpos_t pos;
    errno = 0;

    if (0 != fgetpos(fp, &pos)) {
        char errmsg[BUFFERSIZE];
        if (strerror_r(errno, errmsg, BUFFERSIZE) != 0) {
            /* Handle error */
        }
        printf("Could not get the file position: %s\n", errmsg);
    }
}

```

Linux provides two versions of `strerror_r()`, known as the *XSI-compliant version* and the *GNU-specific version*. This compliant solution assumes the XSI-compliant version, which is the default when an application is compiled as required by POSIX (that is, by defining `_POSIX_C_SOURCE` or `_XOPEN_SOURCE` appropriately). The `strerror_r()` manual page lists versions that are available on a particular system.

Risk Assessment

Race conditions caused by multiple threads invoking the same library function can lead to [abnormal termination](#) of the application, data integrity violations, or a [denial-of-service attack](#).

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
CON33-C	Medium	Probable	High	P4	L3

Related Vulnerabilities

Search for [vulnerabilities](#) resulting from the violation of this rule on the [CERT website](#).

Automated Detection

Tool	Version	Checker	Description
------	---------	---------	-------------

Astrée	19.04		Supported, but no explicit checker
CodeSonar	5.2p0	BADFUNC.RANDOM.RAND BADFUNC.TEMP.TMPNAM BADFUNC.TTYNAME	Use of <code>rand</code> (includes check for uses of <code>srand()</code>) Use of <code>tmpnam</code> (includes check for uses of <code>tmpnam_r()</code>) Use of <code>ttyname</code>
Compass/ROSE			A module written in Compass/ROSE can detect violations of this rule
LDRA tool suite	9.7.1	44 S	Partially Implemented
Parasoft C/C++test	10.4.2	CERT_C-CON33-a	Avoid using thread-unsafe functions
Polyspace Bug Finder	R2019b	CERT C: Rule CON33-C	Checks for data race through standard library function call (rule fully covered)
PRQA QA-C	9.7	5037, 4976, 4977	
PRQA QA-C++	4.4	4976, 4977, 5021	

Related Guidelines

[Key here](#) (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
CERT C Secure Coding Standard	ERR30-C . Set <code>errno</code> to zero before calling a library function known to set <code>errno</code> , and check <code>errno</code> only after the function returns a value indicating failure	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT C	CON00-CPP . Avoid assuming functions are thread safe unless otherwise specified	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-330	2017-06-28: CERT: Partial overlap
CWE 2.11	CWE-377	2017-06-28: CERT: Partial overlap
CWE 2.11	CWE-676	2017-05-18: CERT: Rule subset of CWE

CERT-CWE Mapping Notes

[Key here](#) for mapping notes

CWE-330 and CON33-C

Independent([MSC30-C](#), [MSC32-C](#), [CON33-C](#))

Intersection([CWE-330](#), [CON33-C](#)) =

- Use of `rand()` or `srand()` from multiple threads, introducing a race condition.

[CWE-330](#) – [CON33-C](#) =

- Use of `rand()` or `srand()` without introducing race conditions
- Use of other dangerous functions

[CON33-C](#) – [CWE-330](#) =

- Use of other global functions (besides `rand()` and `srand()`) introducing race conditions

CWE-377 and CON33-C

Intersection(CWE-377, CON33-C) =

- Use of `tmpnam()` from multiple threads, introducing a race condition.

CWE-377 – CON33-C =

- Insecure usage of `tmpnam()` without introducing race conditions
- Insecure usage of other functions for creating temporary files (see CERT recommendation FIO21-C for details)

CON33-C – CWE-377 =

- Use of other global functions (besides `tmpnam()`) introducing race conditions

CWE-676 and CON33-C

- Independent(ENV33-C, CON33-C, STR31-C, EXP33-C, MSC30-C, ERR34-C)
- CON33-C lists standard C library functions that manipulate global data (e.g., `locale()`), that can be dangerous to use in a multithreaded context.
- CWE-676 = Union(CON33-C, list) where list =
- Invocation of the following functions without introducing a race condition:
 - `rand()`, `srand()`, `getenv()`, `getenv_s()`, `strtok()`, `strerror()`, `asctime()`, `ctime()`, `localtime()`, `gmtime()`, `setlocale()`, `ATOMIC_VAR_INIT`, `atomic_init()`, `tmpnam()`, `mbrtoc16()`, `c16rtomb()`, `mbrtoc32()`, `c32rtomb()`
- Invocation of other dangerous functions

Bibliography

[IEEE Std 1003.1:2013]	Section 2.9.1, "Thread Safety"
[ISO/IEC 9899:2011]	Subclause 7.24.6.2, "The <code>strerror</code> Function"
[Open Group 1997b]	Section 10.12, "Thread-Safe POSIX.1 and C-Language Functions"

