CON31-C. Do not destroy a mutex while it is locked

Mutexes are used to protect shared data structures being concurrently accessed. If a mutex is destroyed while a thread is blocked waiting for that mutex, critical sections and shared data are no longer protected.


> The `mtx_destroy` function releases any resources used by the mutex pointed to by `mtx`. No threads can be blocked waiting for the mutex pointed to by `mtx`.

This statement implies that destroying a mutex while a thread is waiting on it is undefined behavior.

Noncompliant Code Example

This noncompliant code example creates several threads that each invoke the `do_work()` function, passing a unique number as an ID. The `do_work()` function initializes the lock mutex if the argument is 0 and destroys the mutex if the argument is `max_threads - 1`. In all other cases, the `do_work()` function provides normal processing. Each thread, except the final cleanup thread, increments the atomic `completed` variable when it is finished.

Unfortunately, this code contains several race conditions, allowing the mutex to be destroyed before it is unlocked. Additionally, there is no guarantee that `lock` will be initialized before it is passed to `mtx_lock()`. Each of these behaviors is undefined.
Compliant Solution

This compliant solution eliminates the race conditions by initializing the mutex in `main()` before creating the threads and by destroying the mutex in `main()` after joining the threads:
```
#include <stdatomic.h>
#include <stddef.h>
#include <threads.h>

mtx_t lock; /* Atomic so multiple threads can increment safely */
atomic_int completed = ATOMIC_VAR_INIT(0);
enum { max_threads = 5 }

int do_work(void *dummy) {
    if (thrd_success != mtx_lock(&lock)) {
        /* Handle error */
    } /* Access data protected by the lock */
    atomic_fetch_add(&completed, 1);
    if (thrd_success != mtx_unlock(&lock)) {
        /* Handle error */
    }
    return 0;
}

int main(void) {
    thrd_t threads[max_threads];

    if (thrd_success != mtx_init(&lock, mtx_plain)) {
        /* Handle error */
    }
    for (size_t i = 0; i < max_threads; i++) {
        if (thrd_success != thrd_create(&threads[i], do_work, NULL)) {
            /* Handle error */
        }
    }
    for (size_t i = 0; i < max_threads; i++) {
        if (thrd_success != thrd_join(threads[i], 0)) {
            /* Handle error */
        }
    }
    mtx_destroy(&lock);
    return 0;
}
```

### Risk Assessment

Destroying a mutex while it is locked may result in invalid control flow and data corruption.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON31-C</td>
<td>Medium</td>
<td>Probable</td>
<td>High</td>
<td>P4</td>
<td>L3</td>
</tr>
</tbody>
</table>

### Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrée</td>
<td>19.04</td>
<td></td>
<td>Supported, but no explicit checker</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>4961, 4962</td>
<td></td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-CON31-a, CERT_C-CON31-b, CERT_C-CON31-c</td>
<td>Do not destroy another thread's mutex, do not use resources that have been freed, do not free resources using invalid pointers</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rule CON31-C</td>
<td>Checks for destruction of locked mutex (rule fully covered)</td>
</tr>
</tbody>
</table>

### Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the [CERT website](https://www.cert.org).
Related Guidelines

Key here (explains table format and definitions)

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Taxonomy item</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWE 2.11</td>
<td>CWE-667, Improper Locking</td>
<td>2017-07-10: CERT: Rule subset of CWE</td>
</tr>
</tbody>
</table>

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-667 and CON31-C/POS48-C

Intersection( CON31-C, POS48-C) = Ø

CWE-667 = Union, CON31-C, POS48-C, list) where list =

- Locking & Unlocking issues besides unlocking another thread's C mutex or pthread mutex.

Bibliography