CON01-C. Acquire and release synchronization primitives in the same module, at the same level of abstraction

All locking and unlocking of mutexes should be performed in the same module and at the same level of abstraction. Failure to follow this recommendation can lead to some lock or unlock operations not being executed by the multithreaded program as designed, eventually resulting in deadlock, race conditions, or other security vulnerabilities, depending on the mutex type.

A common consequence of improper locking is for a mutex to be unlocked twice, via two calls to `mtx_unlock()`. This can cause the unlock operation to return errors. In the case of recursive mutexes, an error is returned only if the lock count is 0 (making the mutex available to other threads) and a call to `mtx_unlock()` is made.

Noncompliant Code Example

In this noncompliant code example for a simplified multithreaded banking system, imagine an account with a required minimum balance. The code would need to verify that all debit transactions are allowable. Suppose a call is made to `debit()` asking to withdraw funds that would bring `account_balance` below `MIN_BALANCE`, which would result in two calls to `mtx_unlock()`. In this example, because the mutex is defined statically, the mutex type is implementation-defined.

```
#include <threads.h>

enum { MIN_BALANCE = 50 };

int account_balance;
mtx_t mp;

/* Initialize mp */

int verify_balance(int amount) {
    if (account_balance - amount < MIN_BALANCE) {
        /* Handle error condition */
        if (mtx_unlock(&mp) == thrd_error) {
            /* Handle error */
        }
        return -1;
    }
    return 0;
}

void debit(int amount) {
    if (mtx_lock(&mp) == thrd_error) {
        /* Handle error */
    }
    if (verify_balance(amount) == -1) {
        if (mtx_unlock(&mp) == thrd_error) {
            /* Handle error */
        }
        return;
    }
    account_balance -= amount;
    if (mtx_unlock(&mp) == thrd_error) {
        /* Handle error */
    }
}
```

Compliant Solution

This compliant solution unlocks the mutex only in the same module and at the same level of abstraction at which it is locked. This technique ensures that the code will not attempt to unlock the mutex twice.
#include <threads.h>

define { MIN_BALANCE = 50 }

static int account_balance;
static mtx_t mp;
/* Initialize mp */

static int verify_balance(int amount) {
    if (account_balance - amount < MIN_BALANCE) {
        return -1;   /* Indicate error to caller */
    }
    return 0;  /* Indicate success to caller */
}

int debit(int amount) {
    if (mtx_lock(&mp) == thrd_error) {
        return -1;   /* Indicate error to caller */
    }
    if (verify_balance(amount)) {
        mtx_unlock(&mp);
        return -1;   /* Indicate error to caller */
    }
    account_balance -= amount;
    if (mtx_unlock(&mp) == thrd_error) {
        return -1;   /* Indicate error to caller */
    }
    return 0;   /* Indicate success */
}

Risk Assessment

Improper use of mutexes can result in denial-of-service attacks or the unexpected termination of a multithreaded program.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON01-C</td>
<td>Low</td>
<td>Probable</td>
<td>Medium</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>CONCURRENCY.LOCK.NOLOCK</td>
<td>Supported, but no explicit checker</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>CONCURRENCY.LOCK.NOLOCK, CONCURRENCY.LOCK.NOUNLOCK</td>
<td>Missing Lock Acquisition, Missing Lock Release</td>
</tr>
<tr>
<td>Coverity</td>
<td>6.5</td>
<td>LOCK</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-CON01-a</td>
<td>Do not abandon unreleased locks</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rec. CON01-C</td>
<td>Checks for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* Missing lock and unlock functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>* Double lock or double unlock</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rec. fully covered</td>
</tr>
</tbody>
</table>

Bibliography
IEEE Std 1003.1-2013

XSH, System Interfaces, pthread_mutex_init
XSH, System Interfaces, pthread_mutex_lock, pthread_mutex_unlock
XSH, System Interfaces, pthread_mutexattr_init