# CON50-CPP. Do not destroy a mutex while it is locked

Mutex objects are used to protect shared data from being concurrently accessed. If a mutex object is destroyed while a thread is blocked waiting for the lock, critical sections and shared data are no longer protected.

The C++ Standard, [thread.mutex.class], paragraph 5 [ISO/IEC 14882-2014], states the following:

> *The behavior of a program is undefined if it destroys a `mutex` object owned by any thread or a thread terminates while owning a `mutex` object.*

Similar wording exists for `std::recursive_mutex`, `std::timed_mutex`, `std::recursive_timed_mutex`, and `std::shared_timed_mutex`. These statements imply that destroying a mutex object 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.

Unfortunately, this code contains a race condition, allowing the mutex to be destroyed while it is still owned, because `start_threads()` may invoke the mutex's destructor before all of the threads have exited.

```
#include <mutex>
#include <thread>

const size_t maxThreads = 10;

void do_work(size_t i, std::mutex *pm) {
  std::lock_guard<std::mutex> lk(*pm);

  // Access data protected by the lock.
}

void start_threads() {
  std::thread threads[maxThreads];
  std::mutex m;

  for (size_t i = 0; i < maxThreads; ++i) {
    threads[i] = std::thread(do_work, i, &m);
  }
}
```

## Compliant Solution

This compliant solution eliminates the race condition by extending the lifetime of the mutex.

```
#include <mutex>
#include <thread>

const size_t maxThreads = 10;

void do_work(size_t i, std::mutex *pm) {
  std::lock_guard<std::mutex> lk(*pm);

  // Access data protected by the lock.
}

std::mutex m;

void start_threads() {
  std::thread threads[maxThreads];

  for (size_t i = 0; i < maxThreads; ++i) {
    threads[i] = std::thread(do_work, i, &m);
  }
}
```

## Compliant Solution

This compliant solution eliminates the race condition by joining the threads before the mutex's destructor is invoked.

```
#include <mutex>
#include <thread>

const size_t maxThreads = 10;

void do_work(size_t i, std::mutex *pm) {
  std::lock_guard<std::mutex> lk(*pm);

  // Access data protected by the lock.
}
void run_threads() {
  std::thread threads[maxThreads];
  std::mutex m;

  for (size_t i = 0; i < maxThreads; ++i) {
    threads[i] = std::thread(do_work, i, &m);
  }

  for (size_t i = 0; i < maxThreads; ++i) {
    threads[i].join();
  }
}
```

## Risk Assessment

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
|---|---|---|---|---|---|
| CON50-CPP | Medium | Probable | High | **P4** | **L3** |

## Automated Detection

| Tool | Version | Checker | Description |
|---|---|---|---|
| Parasoft C/C++test | 10.4.2 | **CERT_CPP-CON50-a** | Do not destroy another thread's mutex |
| Polyspace Bug Finder | R2019b | CERT C++: CON50-CPP | Checks for destruction of locked mutex (rule partially covered) |
| PRQA QA-C++ | 4.4 | **4961, 4962** | |

## Related Vulnerabilities

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

## Related Guidelines

| | |
|---|---|
| MITRE CWE | CWE-667, Improper Locking |
| SEI CERT C Coding Standard | CON31-C. Do not destroy a mutex while it is locked |

## Bibliography


| | |
|---|---|
| [ISO/IEC 14882-2014] | Subclause 30.4.1, "Mutex Requirements" |