POS48-C. Do not unlock or destroy another POSIX thread's mutex

Mutexes are used to protect shared data structures being accessed concurrently. The thread that locks the mutex owns it, and the owning thread should be the only thread to unlock the mutex. If the mutex is destroyed while still in use, critical sections and shared data are no longer protected. This rule is a specific instance of CON31-C. Do not unlock or destroy another thread's mutex using POSIX threads.

Noncompliant Code Example

In this noncompliant code example, a race condition exists between a cleanup and a worker thread. The cleanup thread destroys the lock, which it believes is no longer in use. If there is a heavy load on the system, the worker thread that held the lock can take longer than expected. If the lock is destroyed before the worker thread has completed modifying the shared data, the program may exhibit unexpected behavior.

```
pthread_mutex_t theLock;
int data;

int cleanupAndFinish(void) {
  int result;
  if ((result = pthread_mutex_destroy(&theLock)) != 0) {
    /* Handle error */
  }
  data++;
  return data;
}

void worker(int value) {
  if ((result = pthread_mutex_lock(&theLock)) != 0) {
    /* Handle error */
  }
  data += value;
  if ((result = pthread_mutex_unlock(&theLock)) != 0) {
    /* Handle error */
  }
}
```

Compliant Solution

This compliant solution requires that there is no chance a mutex will be needed after it has been destroyed. As always, it is important to check for error conditions when locking the mutex.

```
mutex_t theLock;
int data;

int cleanupAndFinish(void) {
  int result;

  /* A user-written function that is application-dependent */
  wait_for_all_threads_to_finish();
  if ((result = pthread_mutex_destroy(&theLock)) != 0) {
    /* Handle error */
  }
  data++;
  return data;
}

void worker(int value) {
  int result;
  if ((result = pthread_mutex_lock(&theLock)) != 0) {
    /* Handle error */
  }
  data += value;
  if ((result = pthread_mutex_unlock(&theLock)) != 0) {
    /* Handle error */
  }
}
```

Risk Assessment

The risks of ignoring mutex ownership are similar to the risk of not using mutexes at all, which can result in a violation of data integrity.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
POS48-C	Medium	Probable	High	**P4**	**L3**

Automated Detection

Tool	Version	Checker	Description
Parasoft C/C++test	10.4.2	**CERT_C-POS48-a** **CERT_C-POS48-b**	Do not destroy another thread's mutex Do not release a lock that has not been acquired
Polyspace Bug Finder	R2018a	Destruction of locked mutex	Task tries to destroy a mutex in the locked state
PRQA QA-C	9.5	**4971, 4972, 4981, 4982**	

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
CWE 2.11	CWE-667, Insufficient locking	2017-07-10: CERT: Rule subset of CWE

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

[Open Group 2004]	`pthread_mutex_lock()/pthread_mutex_unlock()`
	`pthread_mutex_destroy()`

← ↑ →