CON04-C. Join or detach threads even if their exit status is unimportant

The `thrd_detach()` function is used to tell the underlying system that resources allocated to a particular thread can be reclaimed once it terminates. This function should be used when a thread's exit status is not required by other threads (and no other thread needs to use `thrd_join()` to wait for it to complete).

Whenever a thread terminates without detaching, the thread's stack is deallocated, but some other resources, including the thread ID and exit status, are left until it is destroyed by either `thrd_join()` or `thrd_detach()`. These resources can be vital for systems with limited resources and can lead to various "resource unavailable" errors, depending on which critical resource gets used up first. For example, if the system has a limit (either per-process or system wide) on the number of thread IDs it can keep track of, failure to release the thread ID of a terminated thread may lead to `thrd_create()` `being unable to` create another thread.

Noncompliant Code Example

This noncompliant code example shows a pool of threads that are not exited correctly:

```
#include <stdio.h>
#include <threads.h>

const size_t thread_no = 5;
const char mess[] = "This is a test";

int message_print(void *ptr){
  const char *msg = (const char *) ptr;
  printf("THREAD: This is the Message %s\n", msg);
  return 0;
}

int main(void){
  /* Create a pool of threads */
  thrd_t thr[thread_no];
  for (size_t i = 0; i < thread_no; ++i) {
    if (thrd_create(&(thr[i]), message_print,
                    (void *)mess) != thrd_success) {
      fprintf(stderr, "Creation of thread %zu failed\n", i);
      /* Handle error */
    }
  }
  printf("MAIN: Thread Message: %s\n", mess);
  return 0;
}
```

Compliant Solution

In this compliant solution, the `message_print()` function is replaced by a similar function that correctly detaches the threads so that the associated resources can be reclaimed on exit:

```
#include <stdio.h>
#include <threads.h>

const size_t thread_no = 5;
const char mess[] = "This is a test";

int message_print(void *ptr){
  const char *msg = (const char *)ptr;
  printf("THREAD: This is the Message %s\n", msg);

  /* Detach the thread, check the return code for errors */
  if (thrd_detach(thrd_current()) != thrd_success) {
    /* Handle error */
  }
  return 0;
}

int main(void) {
  /* Create a pool of threads */
  thrd_t thr[thread_no];
  for(size_t i = 0; i < thread_no; ++i) {
    if (thrd_create(&(thr[i]), message_print,
                    (void *)mess) != thrd_success) {
      fprintf(stderr, "Creation of thread %zu failed\n", i);
      /* Handle error */
    }
  }
  printf("MAIN: Thread Message: %s\n", mess);
  return 0;
}
```

Risk Assessment

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
CON04-C	Low	Unlikely	High	P1	L3

Related Vulnerabilities

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

Bibliography

[IEEE Std 1003.1:2013]	XSH, System Interfaces, `pthread_detach`