WIN01-C. Do not forcibly terminate execution

When a thread terminates under normal conditions, thread-specific resources such as the initial stack space and thread-specific HANDLE objects are released automatically by the system and notifications are sent to other parts of the application, such as DLL_THREAD_DETACH messages being sent to DLLs. However, if a thread is forcibly terminated by calling `TerminateThread()`, the cleanup and notifications do not have the chance to run. MSDN states `TerminateThread` is a dangerous function that should only be used in the most extreme cases. You should call `TerminateThread` only if you know exactly what the target thread is doing, and you control all of the code that the target thread could possibly be running at the time of the termination. For example, `TerminateThread` can result in the following problems:

- If the target thread owns a critical section, the critical section will not be released.
- If the target thread is allocating memory from the heap, the heap lock will not be released.
- If the target thread is executing certain kernel32 calls when it is terminated, the kernel32 state for the thread's process could be inconsistent.
- If the target thread is manipulating the global state of a shared DLL, the state of the DLL could be destroyed, affecting other users of the DLL.

On some platforms (such as Microsoft Windows XP and Microsoft Windows Server 2003), the thread's initial stack is not freed, causing a resource leak. Processes behave similar to threads, and so share the same concerns. Do not use the `TerminateThread()` or `TerminateProcess()` APIs. Instead, you should prefer to exit threads and processes by returning from the entrypoint, by calling `ExitThread()`, or by calling `ExitProcess()`.

Noncompliant Code Example

In the following example, `TerminateThread()` is used to forcibly terminate another thread, which can leak resources and leave the application in an indeterminable state.

```c
#include <Windows.h>

DWORD ThreadID;  /* Filled in by call to CreateThread */

/* Thread 1 */
DWORD WINAPI ThreadProc(LPVOID param) {
    /* Performing work */
}

/* Thread 2 */
HANDLE hThread = OpenThread(THREAD_TERMINATE, FALSE, ThreadID);
if (hThread) {
    TerminateThread(hThread, 0xFF);
    CloseHandle(hThread);
}
```

Compliant Solution

The compliant solution avoids calling `TerminateThread()` by requesting the thread terminate itself by exiting its entrypoint. It does so in a lock-free, thread-safe manner by using the `InterlockedCompareExchange()` and `InterlockedExchange()` Win32 APIs.

```c
#include <Windows.h>

DWORD ThreadID;  /* Filled in by call to CreateThread */
LONG ShouldThreadExit = 0;

/* Thread 1 */
DWORD WINAPI ThreadProc(LPVOID param) {
    while (1) {
        /* Performing work */
        if (1 == InterlockedCompareExchange(&ShouldThreadExit, 0, 1))
            return 0xFF;
    }
}

/* Thread 2 */
InterlockedExchange(&ShouldThreadExit, 1);
```
Risk Assessment

Incorrectly using threads that asynchronously cancel may result in silent corruption, resource leaks, and, in the worst case, unpredictable interactions.

<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>WIN01-C</td>
<td>High</td>
<td>Likely</td>
<td>High</td>
<td>P9</td>
<td>L2</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>bad-function</td>
<td>Partially checked</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>(customization)</td>
<td>Users can add a custom check for all uses of TerminateThread() and TerminateProcess().</td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>bad-function</td>
<td>Partially checked</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

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

Related Guidelines

TODO (notably, should be one about using ExitThread() from C++ code.

Bibliography