POS44-C. Do not use signals to terminate threads

Do not send an uncaught signal to kill a thread because the signal kills the entire process, not just the individual thread. This rule is a specific instance of SIG02-C. Avoid using signals to implement normal functionality.

In POSIX systems, using the \texttt{signal()} function in a multithreaded program falls under exception CON37C-C-EX0 of rule CON37-C. Do not call signal() in a multithreaded program.

Noncompliant Code Example

This code uses the \texttt{pthread_kill()} function to send a \texttt{SIGTERM} signal to the created thread. The thread receives the signal, and the entire process is terminated.

```c
void func(void *foo) {
    /* Execution of thread */
}

int main(void) {
    int result;
    pthread_t thread;

    if ((result = pthread_create(&thread, NULL, func, 0)) != 0) { /* Handle Error */
    }
    if ((result = pthread_kill(thread, SIGTERM)) != 0) { /* Handle Error */
    }

    /* This point is not reached because the process terminates in pthread_kill() */
    return 0;
}
```

Compliant Solution

This compliant code uses instead the \texttt{pthread_cancel()} function to terminate the thread. The thread continues to run until it reaches a cancellation point. See The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 Edition\cite{OpenGroup2004} for lists of functions that are required and allowed to be cancellation points. If the cancellation type is set to asynchronous, the thread is terminated immediately. However, POSIX requires only the \texttt{pthread_cancel()}, \texttt{pthread_setcancelstate()}, and \texttt{pthread_setcanceltype()} functions to be async-cancel safe. An application that calls other POSIX functions with asynchronous cancellation enabled is nonconforming. Consequently, we recommend disallowing asynchronous cancellation, as explained by POS47-C. Do not use threads that can be canceled asynchronously.

```c
void func(void *foo) {
    /* Execution of thread */
}

int main(void) {
    int result;
    pthread_t thread;

    if ((result = pthread_create(&thread, NULL, func, 0)) != 0) { /* Handle Error */
    }
    if ((result = pthread_cancel(thread)) != 0) { /* Handle Error */
    }

    /* Continue executing */
    return 0;
}
```

Risk Assessment

Sending the signal to a process causes it to be abnormally terminated.
Rule | Severity | Likelihood | Remediation Cost | Priority | Level
--- | --- | --- | --- | --- | ---
POS44-C | Low | Probable | Low | P6 | L2

**Automated Detection**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-POS44-a</td>
<td>The signal handling facilities of &lt;signal.h&gt; shall not be used</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rule POS44-C</td>
<td>Checks for use of signal to kill thread (rule fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>5034</td>
<td></td>
</tr>
</tbody>
</table>

**Related Vulnerabilities**

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

**Bibliography**

- [OpenBSD] signal() Man Page
- [MKS] pthread_cancel() Man Page
- [Open Group 2004] Threads Overview