**CON37-C. Do not call signal() in a multithreaded program**

Calling the `signal()` function in a multithreaded program is undefined behavior. (See undefined behavior 135.)

**Noncompliant Code Example**

This noncompliant code example invokes the `signal()` function from a multithreaded program:

```c
#include <signal.h>
#include <stddef.h>
#include <threads.h>

volatile sig_atomic_t flag = 0;

void handler(int signum) {
    flag = 1;
}

/* Runs until user sends SIGUSR1 */
int func(void *data) {
    while (!flag) {
        /* ... */
    }
    return 0;
}

int main(void) {
    signal(SIGUSR1, handler); /* Undefined behavior */
    thrd_t tid;

    if (thrd_success != thrd_create(&tid, func, NULL)) {
        /* Handle error */
    }
    /* ... */
    return 0;
}
```

NOTE: The `SIGUSR1` signal value is not defined in the C Standard; consequently, this is not a C-compliant code example.

**Compliant Solution**

This compliant solution uses an object of type `atomic_bool` to indicate when the child thread should terminate its loop:
```c
#include <stdatomic.h>
#include <stdbool.h>
#include <stddef.h>
#include <threads.h>

atomic_bool flag = ATOMIC_VAR_INIT(false);

int func(void *data) {
    while (!flag) {
        /* ... */
    }
    return 0;
}

int main(void) {
    thrd_t tid;

    if (thrd_success != thrd_create(&tid, func, NULL)) {
        /* Handle error */
    }
    /* ... */
    /* Set flag when done */
    flag = true;
    return 0;
}
```

Exceptions

**CON37-C-EX1:** Implementations such as POSIX that provide defined behavior when multithreaded programs use custom signal handlers are exempt from this rule [IEEE Std 1003.1-2013].

Risk Assessment

Mixing signals and threads causes undefined behavior.

<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>CON37-C</td>
<td>Low</td>
<td>Probable</td>
<td>Low</td>
<td>P6</td>
<td>L2</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

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

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>stlilb-use-signal</td>
<td>Fully checked</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>BADFUNC.SIGNAL</td>
<td>Use of signal</td>
</tr>
<tr>
<td>Coverity</td>
<td>2017.07</td>
<td>MISRA C 2012 Rule 21.5</td>
<td>Over-constraining</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>44 S</td>
<td>Enhanced enforcement</td>
</tr>
<tr>
<td>Parasoft C/C++-test</td>
<td>10.4.2</td>
<td>CERT_C-CON37-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 CON37-C</td>
<td>Checks for signal call in multithreaded program (rule fully covered)</td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>stlilb-use-signal</td>
<td>Fully checked</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>5021</td>
<td></td>
</tr>
<tr>
<td>PRQA QA-C++</td>
<td>4.4</td>
<td>5022</td>
<td></td>
</tr>
</tbody>
</table>

Bibliography