PRE32-C. Do not use preprocessor directives in invocations of function-like macros

The arguments to a macro must not include preprocessor directives, such as `#define`, `#ifdef`, and `#include`. Doing so results in undefined behavior, according to the C Standard, 6.10.3, paragraph 11 [ISO/IEC 9899:2011]:

> The sequence of preprocessing tokens bounded by the outside-most matching parentheses forms the list of arguments for the function-like macro. The individual arguments within the list are separated by comma preprocessing tokens, but comma preprocessing tokens between matching inner parentheses do not separate arguments. If there are sequences of preprocessing tokens within the list of arguments that would otherwise act as preprocessing directives, the behavior is undefined.

See also undefined behavior 93.

This rule also applies to the use of preprocessor directives in arguments to a function where it is unknown whether or not the function is implemented using a macro. For example, standard library functions, such as `memcpy()`, `printf()`, and `assert()`, may be implemented as macros.

Noncompliant Code Example

In this noncompliant code example [GCC Bugs], the programmer uses preprocessor directives to specify platform-specific arguments to `memcpy()`. However, if `memcpy()` is implemented using a macro, the code results in undefined behavior.

```c
#include <string.h>

void func(const char *src) {
    /* Validate the source string; calculate size */
    char *dest;
    /* malloc() destination string */
    memcpy(dest, src,
    #ifdef PLATFORM1
        12
    #else
        24
    #endif
    );
    /* ... */
}
```

Compliant Solution

In this compliant solution [GCC Bugs], the appropriate call to `memcpy()` is determined outside the function call:

```c
#include <string.h>

void func(const char *src) {
    /* Validate the source string; calculate size */
    char *dest;
    /* malloc() destination string */
    #ifdef PLATFORM1
        memcpy(dest, src, 12);
    #else
        memcpy(dest, src, 24);
    #endif
    /* ... */
}
```

Risk Assessment

Including preprocessor directives in macro arguments is 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>PRE32-C</td>
<td>Low</td>
<td>Unlikely</td>
<td>Medium</td>
<td>P2</td>
<td>L3</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>macro-argument-hash</td>
<td>Fully checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-PRE32</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.1p0</td>
<td>LANG.PREPROC.MACROARG</td>
<td>Preprocessing directives in macro argument</td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.PRE32</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>341 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-PRE32-a</td>
<td>Arguments to a function-like macro shall not contain tokens that look like preprocessing directives</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019a</td>
<td>CERT: Rule PRE32-C</td>
<td>Checks for preprocessor directive in macro argument (rule fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.5</td>
<td>0853</td>
<td></td>
</tr>
<tr>
<td>PRQA QA-C++</td>
<td>4.3</td>
<td>1072</td>
<td></td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>macro-argument-hash</td>
<td>Fully checked</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

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

Bibliography

<table>
<thead>
<tr>
<th>[GCC Bugs]</th>
<th>&quot;Non-bugs&quot;</th>
</tr>
</thead>
</table>