ARR36-C. Do not subtract or compare two pointers that do not refer to the same array

When two pointers are subtracted, both must point to elements of the same array object or just one past the last element of the array object (C Standard, 6.5.6 [ISO/IEC 9899:2011]); the result is the difference of the subscripts of the two array elements. Otherwise, the operation is undefined behavior. (See undefined behavior 48.)

Similarly, comparing pointers using the relational operators <, <=, >, and >= gives the positions of the pointers relative to each other. Subtracting or comparing pointers that do not refer to the same array is undefined behavior. (See undefined behavior 48 and undefined behavior 53.)

Comparing pointers using the equality operators == and != has well-defined semantics regardless of whether or not either of the pointers is null, points into the same object, or points one past the last element of an array object or function.

Noncompliant Code Example

In this noncompliant code example, pointer subtraction is used to determine how many free elements are left in the nums array:

```c
#include <stddef.h>
enum { SIZE = 32 };
void func(void) {
    int nums[SIZE];
    int end;
    int *next_num_ptr = nums;
    size_t free_elements;

    /* Increment next_num_ptr as array fills */

    free_elements = &end - next_num_ptr;
}
```

This program incorrectly assumes that the nums array is adjacent to the end variable in memory. A compiler is permitted to insert padding bits between these two variables or even reorder them in memory.

Compliant Solution

In this compliant solution, the number of free elements is computed by subtracting next_num_ptr from the address of the pointer past the nums array. While this pointer may not be dereferenced, it may be used in pointer arithmetic.

```c
#include <stddef.h>
enum { SIZE = 32 };
void func(void) {
    int nums[SIZE];
    int *next_num_ptr = nums;
    size_t free_elements;

    /* Increment next_num_ptr as array fills */

    free_elements = &(nums[SIZE]) - next_num_ptr;
}
```

Exceptions

ARR36-C-EX1: Comparing two pointers to distinct members of the same struct object is allowed. Pointers to structure members declared later in the structure compare greater-than pointers to members declared earlier in the structure.

Risk Assessment

<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>ARR36-C</td>
<td>Medium</td>
<td>Probable</td>
<td>Medium</td>
<td>P8</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>pointer-subtraction</td>
<td>Partially checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-ARR36</td>
<td>Can detect operations on pointers that are unrelated</td>
</tr>
<tr>
<td>Coverity</td>
<td>2017.07</td>
<td>MISRA C 2004 17.2</td>
<td>Implemented</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MISRA C 2004 17.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MISRA C 2012 18.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MISRA C 2012 18.3</td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>437 S, 438 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parosoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-ARR36-a</td>
<td>Pointer arithmetic shall only be applied to pointers that address an array or array element</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rule ARR36-C</td>
<td>Checks for subtraction or comparison between pointers to different arrays (rule partially covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>0487, 0513, 2668, 2669, 2761, 2762, 2763, 2766, 2767, 2768, 2771, 2772, 2773</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>6.23</td>
<td>V736, V782</td>
<td></td>
</tr>
<tr>
<td>TrustInSoft Analyzer</td>
<td>1.38</td>
<td>differing_blocks</td>
<td>Exhaustively verified (see the compliant and the non-compliant example).</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

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

Related Guidelines

Key here (explains table format and definitions)

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Taxonomy item</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C</td>
<td>CTR54-CPP. Do not subtract iterators that do not refer to the same container</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>ISO/IEC TS 17961</td>
<td>Subtracting or comparing two pointers that do not refer to the same array [probb]</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>CWE 2.11</td>
<td>CWE-469, Use of Pointer Subtraction to Determine Size</td>
<td>2017-07-10: CERT: Exact</td>
</tr>
<tr>
<td>CWE 3.11</td>
<td>CWE-469, Use of Pointer Subtraction to Determine Size</td>
<td>2018-10-18:CERT:CWE subset of rule</td>
</tr>
</tbody>
</table>

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-469 and ARR36-C

CWE-469 = Subset(ARR36-C)

ARR36-C = Union(CWE-469, list) where list =
- Pointer comparisons using the relational operators <, <=, >, and >, where the pointers do not refer to the same array

Bibliography

[Banahan 2003] Section 5.3, "Pointers"  
Section 5.7, "Expressions Involving Pointers"
