ARR01-C. Do not apply the sizeof operator to a pointer when taking the size of an array

The `sizeof` operator yields the size (in bytes) of its operand, which can be an expression or the parenthesized name of a type. However, using the `sizeof` operator to determine the size of arrays is error prone.

The `sizeof` operator is often used in determining how much memory to allocate via `malloc()`. However, using an incorrect size is a violation of MEM35-C. Allocate sufficient memory for an object.

Noncompliant Code Example

In this noncompliant code example, the function `clear()` zeros the elements in an array. The function has one parameter declared as `int array[]` and is passed a static array consisting of 12 `int`s as the argument. The function `clear()` uses the idiom `sizeof(array) / sizeof(array[0])` to determine the number of elements in the array. However, `array` has a pointer type because it is a parameter. As a result, `sizeof(array)` is equal to the `array` `sizeof(array)`.

```c
void clear(int array[]) {
    for (size_t i = 0; i < sizeof(array) / sizeof(array[0]); ++i) {
        array[i] = 0;
    }
}

void dowork(void) {
    int dis[12];
    clear(dis);
    /* ... */
}
```

Footnote 103 in subclause 6.5.3.4 of the C Standard [ISO/IEC 9899:2011] applies to all array parameters:

> When applied to a parameter declared to have array or function type, the `sizeof` operator yields the size of the adjusted (pointer) type.

Compliant Solution

In this compliant solution, the size of the array is determined inside the block in which it is declared and passed as an argument to the function:

```c
void clear(int array[], size_t len) {
    for (size_t i = 0; i < len; i++) {
        array[i] = 0;
    }
}

void dowork(void) {
    int dis[12];
    clear(dis, sizeof(dis) / sizeof(dis[0]));
    /* ... */
}
```

This `sizeof(array) / sizeof(array[0])` idiom will succeed provided the original definition of `array` is visible.

Noncompliant Code Example

In this noncompliant code example, `sizeof(a)` does not equal `100 * sizeof(int)`, because the `sizeof` operator, when applied to a parameter declared to have array type, yields the size of the adjusted (pointer) type even if the parameter declaration specifies a length:
enum {ARR_LEN = 100};

void clear(int a[ARR_LEN]) {
    memset(a, 0, sizeof(a)); /* Error */
}

int main(void) {
    int b[ARR_LEN];
    clear(b);
    assert(b[ARR_LEN / 2]==0); /* May fail */
    return 0;
}

Compliant Solution

In this compliant solution, the size is specified using the expression len * sizeof(int):

enum {ARR_LEN = 100};

void clear(int a[], size_t len) {
    memset(a, 0, len * sizeof(int));
}

int main(void) {
    int b[ARR_LEN];
    clear(b, ARR_LEN);
    assert(b[ARR_LEN / 2]==0); /* Cannot fail */
    return 0;
}

Risk Assessment

Incorrectly using the sizeof operator to determine the size of an array can result in a buffer overflow, allowing the execution of arbitrary code.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARR01-C</td>
<td>High</td>
<td>Probable</td>
<td>Low</td>
<td>P18</td>
<td>L1</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>sizeof-array-parameter</td>
<td>Fully checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-ARR01</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td></td>
<td></td>
<td>Can detect violations of the recommendation but cannot distinguish between incomplete array declarations and pointer declarations</td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>CWARN.MEMSET.SIZEOF.PTR</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>401 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-ARR01-a</td>
<td>Do not call 'sizeof' on a pointer type</td>
</tr>
</tbody>
</table>
| Polyspace Bug Finder | R2019b  | CERT C: Rec. ARR01-C  | Checks for:
|                      |         |                       |   • Wrong type used in sizeof
|                      |         |                       |   • Possible misuse of sizeof
|                      |         |                       |   Rec, fully covered.                                                        |
| Splint                | 3.1.1   |                       |                                                                             |
| PVS-Studio            | 7.07    | VS11, VS12, VS14, VS669, VS79, V604, V697 |                                                                   |
| RuleChecker           | 19.04   | sizeof-array-parameter| Fully checked                                                               |
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>CTR01-CPP. Do not apply the sizeof operator to a pointer when taking the size of an array</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>CWE 2.11</td>
<td>CWE-467, Use of sizeof() on a pointer type</td>
<td>Prior to 2018-01-12: CERT:</td>
</tr>
<tr>
<td>ISO/IEC TS 17961</td>
<td>Taking the size of a pointer to determine the size of the pointed-to type [sizeofptr]</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>MITRE CWE</td>
<td>CWE-569</td>
<td>Prior to 2018-01-12:</td>
</tr>
<tr>
<td>MITRE CWE</td>
<td>CWE-783</td>
<td>Prior to 2018-01-12:</td>
</tr>
</tbody>
</table>

Bibliography

[Drepper 2006] Section 2.1.1, "Respecting Memory Bounds"

[ISO/IEC 9899:2011] Subclause 6.5.3.4, "The sizeof and __Alignof Operators"