MEM07-C. Ensure that the arguments to calloc(), when multiplied, do not wrap

The calloc() function takes two arguments: the number of elements to allocate and the storage size of those elements. Typically, calloc() implementations multiply these arguments to determine how much memory to allocate. Historically, some implementations failed to check whether out-of-bounds results silently wrapped [RUS-CERT Advisory 2002-08-02]. If the result of multiplying the number of elements to allocate and the storage size wraps, less memory is allocated than was requested. As a result, it is necessary to ensure that these arguments, when multiplied, do not wrap.

Modern implementations of the C standard library should check for wrap. If the calloc() function implemented by the libraries used for a particular implementation properly handles unsigned integer wrapping (in conformance with INT30-C. Ensure that unsigned integer operations do not wrap) when multiplying the number of elements to allocate and the storage size, that is sufficient to comply with this recommendation and no further action is required.

Noncompliant Code Example

In this noncompliant example, the user-defined function get_size() (not shown) is used to calculate the size requirements for a dynamic array of long int that is assigned to the variable num_elements. When calloc() is called to allocate the buffer, num_elements is multiplied by sizeof(long) to compute the overall size requirements. If the number of elements multiplied by the size cannot be represented as a size_t, then calloc() may allocate a buffer of insufficient size. When data is copied to that buffer, an overflow may occur.

```c
size_t num_elements;
long *buffer = (long *)calloc(num_elements, sizeof(long));
if (buffer == NULL) {
    /* Handle error condition */
} /* ... */
free(buffer);
buffer = NULL;
```

Compliant Solution

In this compliant solution, the two arguments num_elements and sizeof(long) are checked before the call to calloc() to determine if wrapping will occur:

```c
long *buffer;
size_t num_elements;
if (num_elements > SIZE_MAX/sizeof(long)) {
    /* Handle error condition */
}
buffer = (long *)calloc(num_elements, sizeof(long));
if (buffer == NULL) {
    /* Handle error condition */
}
```

Note that the maximum amount of allocatable memory is typically limited to a value less than SIZE_MAX (the maximum value of size_t). Always check the return value from a call to any memory allocation function in compliance with ERR33-C. Detect and handle standard library errors.

Risk Assessment

Unsigned integer wrapping in memory allocation functions can lead to buffer overflows that can be exploited by an attacker to execute arbitrary code with the permissions of the vulnerable process. Most implementations of calloc() now check to make sure silent wrapping does not occur, but it is not always safe to assume the version of calloc() being used is secure, particularly when using dynamically linked libraries.

<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>MEM07-C</td>
<td>High</td>
<td>Unlikely</td>
<td>Medium</td>
<td>P6</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></td>
<td>Supported, but no explicit checker</td>
</tr>
</tbody>
</table>
### Related Vulnerabilities

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

### Related Guidelines

<table>
<thead>
<tr>
<th>SEI CERT C++ Coding Standard</th>
<th>VOID MEM07-CPP. Ensure that the arguments to <code>calloc()</code>, when multiplied, can be represented as a <code>size_t</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>MITRE CWE</td>
<td><code>CWE-190</code>, Integer overflow (wrap or wraparound)</td>
</tr>
<tr>
<td></td>
<td><code>CWE-128</code>, Wrap-around error</td>
</tr>
</tbody>
</table>

### Bibliography

- **[RUS-CERT]** Advisory 2002-08:02, "Flaw in `calloc` and Similar Routines"
- **[Seacord 2013]** Chapter 4, "Dynamic Memory Management"
- **[Secunia]** Advisory SA10635, "HP-UX `calloc` Buffer Size Miscalculation Vulnerability"