MEM36-C. Do not modify the alignment of objects by calling realloc()

Do not invoke realloc() to modify the size of allocated objects that have stricter alignment requirements than those guaranteed by malloc(). Storage allocated by a call to the standard aligned_alloc() function, for example, can have stricter than normal alignment requirements. The C standard requires only that a pointer returned by realloc() be suitably aligned so that it may be assigned to a pointer to any type of object with a fundamental alignment requirement.

Noncompliant Code Example

This noncompliant code example returns a pointer to allocated memory that has been aligned to a 4096-byte boundary. If the resize argument to the realloc() function is larger than the object referenced by ptr, then realloc() will allocate new memory that is suitably aligned so that it may be assigned to a pointer to any type of object with a fundamental alignment requirement but may not preserve the stricter alignment of the original object.

```c
#include <stdlib.h>

void func(void) {
    size_t resize = 1024;
    size_t alignment = 1 << 12;
    int *ptr;
    int *ptr1;

    if (NULL == (ptr = (int *)aligned_alloc(alignment, sizeof(int)))) {
        /* Handle error */
    }

    if (NULL == (ptr1 = (int *)realloc(ptr, resize))) {
        /* Handle error */
    }
}
```

Implementation Details

When compiled with GCC 4.1.2 and run on the x86_64 Red Hat Linux platform, the following code produces the following output:

```c
#include <stdlib.h>
#include <stdio.h>

int main(void) {
    size_t  size = 16;
    size_t resize = 1024;
    size_t align = 1 << 12;
    int *ptr;
    int *ptr1;

    if (posix_memalign((void **)&ptr, align , size) != 0) {
        exit(EXIT_FAILURE);
    }

    printf("memory aligned to %zu bytes\n", align);
    printf("ptr = %p\n", ptr);

    if ((ptr1 = (int *) realloc((int *)ptr, resize)) == NULL) {
        exit(EXIT_FAILURE);
    }

    puts("After realloc(): 
");
    printf("ptr1 = %p\n", ptr1);
    free(ptr1);
    return 0;
}
```
OUTPUT

memory aligned to 4096 bytes
ptr = 0x1621b000

After realloc():
ptr1 = 0x1621a010

ptr1 is no longer aligned to 4096 bytes.

Compliant Solution

This compliant solution allocates `resize` bytes of new memory with the same alignment as the old memory, copies the original memory content, and then frees the old memory. This solution has implementation-defined behavior because it depends on whether extended alignments in excess of `__Alignof (max_align_t)` are supported and the contexts in which they are supported. If not supported, the behavior of this compliant solution is undefined.

```c
#include <stdlib.h>
#include <string.h>

void func(void) {
    size_t resize = 1024;
    size_t alignment = 1 << 12;
    int *ptr;
    int *ptr1;

    if (NULL == (ptr = (int *)aligned_alloc(alignment,
                                             sizeof(int)))) {
        /* Handle error */
    }

    if (NULL == (ptr1 = (int *)aligned_alloc(alignment,
                                              resize))) {
        /* Handle error */
    }

    if (NULL == (memcpy(ptr1, ptr, sizeof(int)))) {
        /* Handle error */
    }

    free(ptr);
}
```

Compliant Solution (Windows)

Windows defines the `_aligned_malloc()` function to allocate memory on a specified alignment boundary. The `_aligned_realloc()` [MSDN] can be used to change the size of this memory. This compliant solution demonstrates one such usage:
```
#include <malloc.h>

void func(void) {
    size_t alignment = 1 << 12;
    int *ptr;
    int *ptr1;

    /* Original allocation */
    if (NULL == (ptr = (int *)_aligned_malloc(sizeof(int),
                                           alignment))) {
        /* Handle error */
    }

    /* Reallocation */
    if (NULL == (ptr1 = (int *)_aligned_realloc(ptr, 1024,
                                              alignment))) {
        _aligned_free(ptr);
        /* Handle error */
    }

    _aligned_free(ptr1);
}
```

The size and alignment arguments for _aligned_malloc() are provided in reverse order of the C Standard aligned_alloc() function.

**Risk Assessment**

Improper alignment can lead to arbitrary memory locations being accessed and written to.

<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>MEM36-C</td>
<td>Low</td>
<td>Probable</td>
<td>High</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></td>
<td>Supported, but no explicit checker</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-MEM36</td>
<td>Fully implemented</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-MEM36-a</td>
<td>Do not modify the alignment of objects by calling realloc()</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rule MEM36-C</td>
<td>Checks for alignment change after memory allocation (rule fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>5027</td>
<td></td>
</tr>
<tr>
<td>PRQA QA-C++</td>
<td>4.4</td>
<td>5034</td>
<td></td>
</tr>
</tbody>
</table>

**Related Vulnerabilities**

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

**Bibliography**

- [ISO/IEC 9899:2011](#) 7.22.3.1, "The aligned_alloc Function"
- [MSDN](#) aligned_malloc()