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.

```
#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:

CODE

```
#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\n", ptr);

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

  puts("After realloc(): \n");
  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.

```
#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.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
MEM36-C	Low	Probable	High	**P2**	**L3**

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04		Supported, but no explicit checker
Axivion Bauhaus Suite	6.9.0	**CertC-MEM36**	Fully implemented
LDRA tool suite	9.7.1	**44 S**	Enhanced enforcement
Parasoft C/C++test	10.4.2	**CERT_C-MEM36-a**	Do not modify the alignment of objects by calling realloc()
PRQA QA-C	9.5	**5027**	
Polyspace Bug Finder	R2019a	CERT C: Rule MEM36-C	Checks for alignment change after memory allocation (rule fully covered)

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()`

← ↑ →