MEM34-C. Only free memory allocated dynamically

The C Standard, Annex J [ISO/IEC 9899:2011], states that the behavior of a program is undefined when

> *The pointer argument to the `free` or `realloc` function does not match a pointer earlier returned by a memory management function, or the space has been deallocated by a call to `free` or `realloc`.*

See also undefined behavior 179.

Freeing memory that is not allocated dynamically can result in heap corruption and other serious errors. Do not call `free()` on a pointer other than one returned by a standard memory allocation function, such as `malloc()`, `calloc()`, `realloc()`, or `aligned_alloc()`.

A similar situation arises when `realloc()` is supplied a pointer to non-dynamically allocated memory. The `realloc()` function is used to resize a block of dynamic memory. If `realloc()` is supplied a pointer to memory not allocated by a standard memory allocation function, the behavior is undefined. One consequence is that the program may terminate abnormally.

This rule does not apply to null pointers. The C Standard guarantees that if `free()` is passed a null pointer, no action occurs.

Noncompliant Code Example

This noncompliant code example sets `c_str` to reference either dynamically allocated memory or a statically allocated string literal depending on the value of `argc`. In either case, `c_str` is passed as an argument to `free()`. If anything other than dynamically allocated memory is referenced by `c_str`, the call to `free(c_str)` is erroneous.

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

enum { MAX_ALLOCATION = 1000 };

int main(int argc, const char *argv[]) {
  char *c_str = NULL;
  size_t len;

  if (argc == 2) {
    len = strlen(argv[1]) + 1;
    if (len > MAX_ALLOCATION) {
      /* Handle error */
    }
    c_str = (char *)malloc(len);
    if (c_str == NULL) {
      /* Handle error */
    }
    strcpy(c_str, argv[1]);
  } else {
    c_str = "usage: $>a.exe [string]";
    printf("%s\n", c_str);
  }
  free(c_str);
  return 0;
}
```

Compliant Solution

This compliant solution eliminates the possibility of `c_str` referencing memory that is not allocated dynamically when passed to `free()`:

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

enum { MAX_ALLOCATION = 1000 };

int main(int argc, const char *argv[]) {
  char *c_str = NULL;
  size_t len;

  if (argc == 2) {
    len = strlen(argv[1]) + 1;
    if (len > MAX_ALLOCATION) {
      /* Handle error */
    }
    c_str = (char *)malloc(len);
    if (c_str == NULL) {
      /* Handle error */
    }
    strcpy(c_str, argv[1]);
  } else {
    printf("%s\n", "usage: $>a.exe [string]");
    return EXIT_FAILURE;
  }
  free(c_str);
  return 0;
}
```

Noncompliant Code Example (`realloc()`)

In this noncompliant example, the pointer parameter to `realloc()`, `buf`, does not refer to dynamically allocated memory:

```
#include <stdlib.h>

enum { BUFSIZE = 256 };

void f(void) {
  char buf[BUFSIZE];
  char *p = (char *)realloc(buf, 2 * BUFSIZE);
  if (p == NULL) {
    /* Handle error */
  }
}
```

Compliant Solution (`realloc()`)

In this compliant solution, `buf` refers to dynamically allocated memory:

```
#include <stdlib.h>

enum { BUFSIZE = 256 };

void f(void) {
  char *buf = (char *)malloc(BUFSIZE * sizeof(char));
  char *p = (char *)realloc(buf, 2 * BUFSIZE);
  if (p == NULL) {
    /* Handle error */
  }
}
```

Note that `realloc()` will behave properly even if `malloc()` failed, because when given a null pointer, `realloc()` behaves like a call to `malloc()`.

Risk Assessment

The consequences of this error depend on the implementation, but they range from nothing to arbitrary code execution if that memory is reused by `malloc()`.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
MEM34-C	High	Likely	Medium	P18	L1

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**invalid-free**	Fully checked
Axivion Bauhaus Suite	6.9.0	**CertC-MEM34**	Can detect memory deallocations for stack objects
Clang	3.9	**clang-analyzer-unix.Malloc**	Checked by `clang-tidy`; can detect some instances of this rule, but does not detect all
CodeSonar	5.2p0	**ALLOC.FNH**	Free non-heap variable
Compass/ROSE			Can detect some violations of this rule
Coverity	2017.07	**BAD_FREE**	Identifies calls to `free()` where the argument is a pointer to a function or an array. It also detects the cases where `free()` is used on an address-of expression, which can never be heap allocated. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary
Klocwork	2018	**FNH.MIGHT** **FNH.MUST** **FUM.GEN.MIGHT** **FUM.GEN.MUST**	
LDRA tool suite	9.7.1	**407 S, 483 S, 644 S, 645 S, 125 D**	Partially implemented
Parasoft C/C++test	10.4.2	**CERT_C-MEM34-a**	Do not free resources using invalid pointers
Parasoft Insure++			Runtime analysis
Polyspace Bug Finder	R2019b	CERT C: Rule MEM34-C	Checks for invalid free of pointer (rule partially covered)
PRQA QA-C	9.7	**2721, 2722, 2723**	
PRQA QA-C++	4.4	**2721**, **2722, 2723**	
PVS-Studio	6.23	**V585**, **V726**	
TrustInSoft Analyzer	1.38	**unclassified ("free expects a free-able address")**	Exhaustively verified (see one compliant and one non-compliant example).

Related Vulnerabilities

CVE-2015-0240 describes a vulnerability in which an uninitialized pointer is passed to `TALLOC_FREE()`, which is a Samba-specific memory deallocation macro that wraps the `talloc_free()` function. The implementation of `talloc_free()` would access the uninitialized pointer, resulting in a remote exploit.

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

Related Guidelines

Key here (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
CERT C Secure Coding Standard	MEM31-C. Free dynamically allocated memory when no longer needed	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT C	MEM51-CPP. Properly deallocate dynamically allocated resources	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TS 17961	Reallocating or freeing memory that was not dynamically allocated [xfree]	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-590, Free of Memory Not on the Heap	2017-07-10: CERT: Exact

Bibliography

[ISO/IEC 9899:2011]	Subclause J.2, "Undefined Behavior"
[Seacord 2013b]	Chapter 4, "Dynamic Memory Management"