MEM31-C. Free dynamically allocated memory when no longer needed

Before the lifetime of the last pointer that stores the return value of a call to a standard memory allocation function has ended, it must be matched by a call to `free()` with that pointer value.

Noncompliant Code Example

In this noncompliant example, the object allocated by the call to `malloc()` is not freed before the end of the lifetime of the last pointer `text_buffer` referring to the object:

```
#include <stdlib.h>

enum { BUFFER_SIZE = 32 };

int f(void) {
  char *text_buffer = (char *)malloc(BUFFER_SIZE);
  if (text_buffer == NULL) {
    return -1;
  }
  return 0;
}
```

Compliant Solution

In this compliant solution, the pointer is deallocated with a call to `free()`:

```
#include <stdlib.h>

enum { BUFFER_SIZE = 32 };

int f(void) {
  char *text_buffer = (char *)malloc(BUFFER_SIZE);
  if (text_buffer == NULL) {
    return -1;
  }

  free(text_buffer);
  return 0;
}
```

Exceptions

MEM31-C-EX1: Allocated memory does not need to be freed if it is assigned to a pointer with static storage duration whose lifetime is the entire execution of a program. The following code example illustrates a pointer that stores the return value from `malloc()` in a `static` variable:

```
#include <stdlib.h>

enum { BUFFER_SIZE = 32 };

int f(void) {
  static char *text_buffer = NULL;
  if (text_buffer == NULL) {
    text_buffer = (char *)malloc(BUFFER_SIZE);
    if (text_buffer == NULL) {
      return -1;
    }
  }
  return 0;
}
```

Risk Assessment

Failing to free memory can result in the exhaustion of system memory resources, which can lead to a denial-of-service attack.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
MEM31-C	Medium	Probable	Medium	P8	L2

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04		Supported, but no explicit checker
Axivion Bauhaus Suite	6.9.0	**CertC-MEM31**	Can detect dynamically allocated resources that are not freed
CodeSonar	5.1p0	**ALLOC.LEAK**	Leak
Compass/ROSE			
Coverity	2017.07	**RESOURCE_LEAK** **ALLOC_FREE_MISMATCH**	Finds resource leaks from variables that go out of scope while owning a resource
Cppcheck	1.66	**leakReturnValNotUsed**	Doesn't use return value of memory allocation function
Klocwork	2018	**MLK.MIGHT** **MLK.MUST** **MLK.RET.MUST** **MLK.RET.MIGHT**	
LDRA tool suite	9.7.1	**50 D**	Partially implemented
Parasoft C/C++test	10.4.2	**CERT_C-MEM31-a**	Ensure resources are freed
Parasoft Insure++			Runtime analysis
Polyspace Bug Finder	R2019b	CERT C: Rule MEM31-C	Checks for memory leak (rule fully covered)
PRQA QA-C	9.7	**2706, 2707, 2708**	
PRQA QA-C++	4.4	**2706, 2707, 2708, 3337, 3338**	
SonarQube C/C++ Plugin	3.11	**S3584**	
Splint	3.1.1		
TrustInSoft Analyzer	1.38	**malloc**	Exhaustively verified.

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)

Taxonomy	Taxonomy item	Relationship
ISO/IEC TR 24772: 2013	Memory Leak [XYL]	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TS 17961	Failing to close files or free dynamic memory when they are no longer needed [fileclose]	Prior to 2018-01-12: CERT: Unspecified Relationship
CWE 2.11	CWE-401, Improper Release of Memory Before Removing Last Reference ("Memory Leak")	2017-07-05: CERT: Exact
CWE 2.11	CWE-404	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-459	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-771	2017-07-06: CERT: Rule subset of CWE
CWE 2.11	CWE-772	2017-07-06: CERT: Rule subset of CWE

CERT-CWE Mapping Notes

Key here for mapping notes

CWE-404/CWE-459/CWE-771/CWE-772 and FIO42-C/MEM31-C

Intersection(FIO42-C, MEM31-C) = Ø

CWE-404 = CWE-459 = CWE-771 = CWE-772

CWE-404 = Union(FIO42-C, MEM31-C list) where list =

- Failure to free resources besides files or memory chunks, such as mutexes)

Bibliography

[ISO/IEC 9899:2011]	Subclause 7.22.3, "Memory Management Functions"