EXP35-C. Do not modify objects with temporary lifetime

The C11 Standard [ISO/IEC 9899:2011] introduced a new term: *temporary lifetime*. Modifying an object with temporary lifetime is undefined behavior. According to subclause 6.2.4, paragraph 8

> *A non-lvalue expression with structure or union type, where the structure or union contains a member with array type (including, recursively, members of all contained structures and unions) refers to an object with automatic storage duration and* temporary *lifetime. Its lifetime begins when the expression is evaluated and its initial value is the value of the expression. Its lifetime ends when the evaluation of the containing full expression or full declarator ends. Any attempt to modify an object with temporary lifetime results in undefined behavior.*

This definition differs from the C99 Standard (which defines modifying the result of a function call or accessing it after the next sequence point as undefined behavior) because a temporary object's lifetime ends when the evaluation containing the full expression or full declarator ends, so the result of a function call can be accessed. This extension to the lifetime of a temporary also removes a quiet change to C90 and improves compatibility with C++.

C functions may not return arrays; however, functions can return a pointer to an array or a `struct` or `union` that contains arrays. Consequently, if a function call returns by value a `struct` or `union` containing an array, do not modify those arrays within the expression containing the function call. Do not access an array returned by a function after the next sequence point or after the evaluation of the containing full expression or full declarator ends.

Noncompliant Code Example (C99)

This noncompliant code example conforms to the C11 Standard; however, it fails to conform to C99. If compiled with a C99-conforming implementation, this code has undefined behavior because the sequence point preceding the call to `printf()` comes between the call and the access by `printf()` of the string in the returned object.

```
#include <stdio.h>

struct X { char a[8]; };

struct X salutation(void) {
  struct X result = { "Hello" };
  return result;
}

struct X addressee(void) {
  struct X result = { "world" };
  return result;
}

int main(void) {
  printf("%s, %s!\n", salutation().a, addressee().a);
  return 0;
}
```

Compliant Solution

This compliant solution stores the structures returned by the call to `addressee()` before calling the `printf()` function. Consequently, this program conforms to both C99 and C11.

```
#include <stdio.h>

struct X { char a[8]; };

struct X salutation(void) {
  struct X result = { "Hello" };
  return result;
}

struct X addressee(void) {
  struct X result = { "world" };
  return result;
}

int main(void) {
  struct X my_salutation = salutation();
  struct X my_addressee = addressee();

  printf("%s, %s!\n", my_salutation.a, my_addressee.a);
  return 0;
}
```

Noncompliant Code Example

This noncompliant code example attempts to retrieve an array and increment the array's first value. The array is part of a `struct` that is returned by a function call. Consequently, the array has temporary lifetime, and modifying the array is undefined behavior.

```
#include <stdio.h>

struct X { int a[6]; };

struct X addressee(void) {
  struct X result = { { 1, 2, 3, 4, 5, 6 } };
  return result;
}

int main(void) {
  printf("%x", ++(addressee().a[0]));
  return 0;
}
```

Compliant Solution

This compliant solution stores the structure returned by the call to `addressee()` as `my_x` before calling the `printf()` function. When the array is modified, its lifetime is no longer temporary but matches the lifetime of the block in `main()`.

```
#include <stdio.h>

struct X { int a[6]; };

struct X addressee(void) {
  struct X result = { { 1, 2, 3, 4, 5, 6 } };
  return result;
}

int main(void) {
  struct X my_x = addressee();
  printf("%x", ++(my_x.a[0]));
  return 0;
}
```

Risk Assessment

Attempting to modify an array or access it after its lifetime expires may result in erroneous program behavior.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP35-C	Low	Probable	Medium	**P4**	**L3**

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-EXP35**	
LDRA tool suite	9.7.1	**642 S, 42 D, 77 D**	Enhanced Enforcement
Parasoft C/C++test	10.4.2	**CERT_C-EXP35-a**	Do not access an array in the result of a function call
Polyspace Bug Finder	R2019a	CERT-C: Rule EXP35-C	Checks for accesses on objects with temporary lifetime (rule fully covered)
PRQA QA-C	9.5	**0450 [U], 0455 [U], 0459 [U], 0465 [U], 0465 [U]**	
Splint	3.1.1		

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	Dangling References to Stack Frames [DCM]	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TR 24772:2013	Side-effects and Order of Evaluation [SAM]	Prior to 2018-01-12: CERT: Unspecified Relationship

Bibliography

[ISO/IEC 9899:2011]	6.2.4, "Storage Durations of Objects"