EXP47-C. Do not call va_arg with an argument of the incorrect type

The variable arguments passed to a variadic function are accessed by calling the `va_arg()` macro. This macro accepts the `va_list` representing the variable arguments of the function invocation and the type denoting the expected argument type for the argument being retrieved. The macro is typically invoked within a loop, being called once for each expected argument. However, there are no type safety guarantees that the type passed to `va_arg` matches the type passed by the caller, and there are generally no compile-time checks that prevent the macro from being invoked with no argument available to the function call. The C Standard, 7.16.1.1, states [ISO/IEC 9899:2011], in part:

> *If there is no actual next argument, or if* type *is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined, except for the following cases:*
>
> *— one type is a signed integer type, the other type is the corresponding unsigned integer type, and the value is representable in both types;*
>
> *— one type is pointer to `void` and the other is a pointer to a character type.*

Ensure that an invocation of the `va_arg()` macro does not attempt to access an argument that was not passed to the variadic function. Further, the type passed to the `va_arg()` macro must match the type passed to the variadic function after default argument promotions have been applied. Either circumstance results in undefined behavior.

Noncompliant Code Example

This noncompliant code example attempts to read a variadic argument of type `unsigned char` with `va_arg()`. However, when a value of type `unsigned char` is passed to a variadic function, the value undergoes default argument promotions, resulting in a value of type `int` being passed.

```
#include <stdarg.h>
#include <stddef.h>

void func(size_t num_vargs, ...) {
  va_list ap;
  va_start(ap, num_vargs);
  if (num_vargs > 0) {
    unsigned char c = va_arg(ap, unsigned char);
    // ...
  }
  va_end(ap);
}

void f(void) {
  unsigned char c = 0x12;
  func(1, c);
}
```

Compliant Solution

The compliant solution accesses the variadic argument with type `int`, and then casts the resulting value to type `unsigned char`:

```
#include <stdarg.h>
#include <stddef.h>

void func(size_t num_vargs, ...) {
  va_list ap;
  va_start(ap, num_vargs);
  if (num_vargs > 0) {
    unsigned char c = (unsigned char) va_arg(ap, int);
    // ...
  }
  va_end(ap);
}

void f(void) {
  unsigned char c = 0x12;
  func(1, c);
}
```

Noncompliant Code Example

This noncompliant code example assumes that at least one variadic argument is passed to the function, and attempts to read it using the `va_arg()` macro. This pattern arises frequently when a variadic function uses a sentinel value to denote the end of the variable argument list. However, the caller passes no variadic arguments to the function, which results in undefined behavior.

```
#include <stdarg.h>

void func(const char *cp, ...) {
  va_list ap;
  va_start(ap, cp);
  int val = va_arg(ap, int);
  // ...
  va_end(ap);
}

void f(void) {
  func("The only argument");
}
```

Compliant Solution

Standard C provides no mechanism to enable a variadic function to determine how many variadic arguments are actually provided to the function call. That information must be passed in an out-of-band manner. Oftentimes this results in the information being encoded in the initial parameter, as in this compliant solution:

```
#include <stdarg.h>
#include <stddef.h>

void func(size_t num_vargs, const char *cp, ...) {
  va_list ap;
  va_start(ap, cp);
  if (num_vargs > 0) {
    int val = va_arg(ap, int);
    // ...
  }
  va_end(ap);
}

void f(void) {
  func(0, "The only argument");
}
```

Risk Assessment

Incorrect use of `va_arg()` results in undefined behavior that can include accessing stack memory.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP47-C	Medium	Likely	High	P6	L2

Automated Detection

Tool	Version	Checker	Description
Axivion Bauhaus Suite	6.9.0	**CertC-EXP47**	
Clang	3.9	`-Wvarargs`	Can detect some instances of this rule, such as promotable types. Cannot detect mismatched types or incorrect number of variadic arguments.
CodeSonar	5.1p0	**BADMACRO.STDARG_H**	Use of <stdarg.h> feature
LDRA tool suite	9.7.1	**44 S**	Enhanced Enforcement
Parasoft C/C++test	10.4.2	**CERT_C-EXP47-a**	Do not call va_arg with an argument of the incorrect type
Polyspace Bug Finder	R2019b	CERT C: Rule EXP47-C	Checks for: • Incorrect data type passed to va_arg • Too many va_arg calls for current argument list Rule fully covered
TrustInSoft Analyzer	1.38	**unclassified (variadic)**	Exhaustively verified (see one compliant and one non-compliant example).

Related Vulnerabilities

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

Bibliography

[ISO/IEC 9899:2011]	Subclause 7.16, "Variable Arguments `<stdarg.h>`" Subclause 6.5.2.2, "Function calls"