N3677 - Examples of Undefined Behavior for Annex J.2 in C23

by David Svoboda <svoboda2sei.cmu.edu>

The paper was created by the Undefined Behavior Study Group, in hopes of publication. We would therefore request a vote to form an editorial group to adjust this paper for eventual publication as a white paper.

Introduction

For each undefined behavior, this document provides an example of code that demonstrates this undefined behavior.

For most undefined behaviors, the example code strictly conforms to the Standard, except for the single undefined behavior. These code examples build executable code, although popular compilers may issue a warning.

However, there are several undefined behaviors (#6 for one) for which we were unable to generate a code example that produces executable code using current compilers. For these, we imagined that a compiler was extended to support an additional feature of the C language, and we then wrote the code for this hypothetical compiler. These examples each are prefaced with "EXTENDED COMPILABLE EXAMPLE".

Format

The format of this document is is:

- Undefined Behavior Number
- Undefined Behavior Definition (taken from Annex J.2)
- Example(s)

Legal Issues

Bibliography

Many code examples were provided by the following sources:

- TS 17961: ISO/IEC TS 17961. *Information Technology—Programming Languages, Their Environments and System Software Interfaces—C Secure Coding Rules*. Geneva, Switzerland: ISO, 2012.
- CERT C Rule This is the published 2016 snapshot of SEI CERT C Rule. The CERT code examples are reproduced with permission of the owner.

- CERT C Recommendations The CERT code examples are reproduced with permission of the owner.

Examples

1. A "shall" or "shall not" requirement that appears outside of a constraint is violated (Clause 4).

See UB Example #4, which violates a 'shall' statement.

Reviewers: svoboda, UBSG

2. A nonempty source file does not end in a new-line character which is not immediately preceded by a backslash character or ends in a partial preprocessing token or comment (5.1.1.2).

```
// Undefined Behavior
/* Comment with end comment marker unintentionally omitted
security_critical_function();
```

Cite: CERT C Rec MSC04-C 1st NCCE

Reviewers: svoboda

3. Token concatenation produces a character sequence matching the syntax of a universal character name (5.1.1.2).

```
#define assign(uc1, uc2, val) uc1##uc2 = val

void func(void) {
  int \u0401;
  // ...
  assign(\u04, 01, 4);   // Undefined Behavior
  // ...
}
```

Cite: CERT C Rule PRE30-C 1st NCCE 2.1.1

Reviewers: svoboda

4. A program in a hosted environment does not define a function named main using one of the specified forms (5.1.2.3.2).

```
#include <stdio.h>

int main(float argc) {  // Undefined Behavior
  printf("main argument count:%f\n", argc);
  return 0;
}
```

Reviewers: s.maddanimath, svoboda, UBSG, j.myers

5. The execution of a program contains a data race (5.1.2.5).

```c
#include <stdatomic.h>
#include <stdbool.h>

static atomic_bool flag = ATOMIC_VAR_INIT(false);

void init_flag(void) {
  atomic_init(&flag, false);
}

void toggle_flag(void) {
  bool temp_flag = atomic_load(&flag);
  temp_flag = !temp_flag;
  atomic_store(&flag, temp_flag);  // Undefined Behavior, race condition
}

bool get_flag(void) {
  return atomic_load(&flag);       // Undefined Behavior, race condition
}
```

Cite: CERT C Rule CON32-C 1st NCCE 14.3.1, CON40-C 1st NCCE 14.11.1

Reviewers: svoboda

6. A character not in the basic source character set is encountered in a source file, except in an identifier, a character constant, a string literal, a header name, a comment, or a preprocessing token that is never converted to a token (5.2.1).

EXTENDED COMPILABLE EXAMPLE: Consider the following non-standard extension: a compiler that accepts the non-ASCII character: ÷ to act as a division operator. Consequently, in this compiler, the following code would be well-defined, but it is undefined behavior for compilers without this extension (and also a syntax error):

```c
// In UTF-8: the division sign == '÷' == U+00F7
double x = 3 ÷ 7; // Undefined Behavior
```

Reviewers: svoboda

7. An identifier, comment, string literal, character constant, or header name contains an invalid multibyte character or does not begin and end in the initial shift state (5.2.2).

```c
/* In UTF-8 Lowercase n-with-tilde == ñ == U+00F1 == 0xC3 0xB1 == \303 \261
*/
```

```
/* const char ma\303ana[] = "tomorrow"; */ /* invalid UTF-8, missing \261 */
/* Undefined Behavior if built in UTF-8 locale, even inside comment */
```

Reviewers: svoboda, USBG, j.myers, s.maddanimath

8. The same identifier has both internal and external linkage in the same translation unit (6.2.2).

```
int i1 = 10;          // Definition, external linkage
static int i2 = 20;   // Definition, internal linkage
extern int i3 = 30;   // Definition, external linkage
int i4;               // Tentative definition, external linkage
static int i5;        // Tentative definition, internal linkage

int i1;               // Valid tentative definition
int i2;               // Undefined Behavior, linkage disagreement with
previous
int i3;               // Valid tentative definition
int i4;               // Valid tentative definition
int i5;               // Undefined Behavior, linkage disagreement with
previous
```

Note: the following is a different example which is more complex:

```
static int a;
int f(void) {
  int a;
  {
     extern int a; // Undefined Behavior, not internal!
  }
}
```

Cite: CERT C Rule DCL36-C 1st NCCE 3.3.1

Reviewers: svoboda, uecker, j.myers

9. An object is referred to outside of its lifetime (6.2.4).

```
void squirrel_away(char **ptr_param) {
  char local[10];
  // Initialize array ...
  *ptr_param = local;
}

void rodent(void) {
  char *ptr;
  squirrel_away(&ptr);
  // Undefined Behavior if ptr is ever read here
}
```

Cite: CERT C Rule DCL30-C 1st NCCE 3.1.1, 2nd NCCE 3.1.4, 3rd NCCE 3.1.6 Cite: CERT C Rule EXP35-C 1st NCCE 4.5.1, 2nd NCCE 4.5.3

Reviewers: svoboda

10. The value of a pointer to an object whose lifetime has ended is used (6.2.4).

```
const char *p;
void dont_do_this(void) {
  const char c_str[] = "This will change";
  p = c_str; // Undefined Behavior if p subsequently read
}

void innocuous(void) {
  const char c_str[] = "Surprise, surprise";
  puts(c_str);
}

int main(void) {
  dont_do_this();
  innocuous();
  puts(p);  // On some platforms, this will print "Surprise, surprise"
  return EXIT_SUCCESS;
}
```

Cite: TS17961 5.14 [nullref] EXAMPLE 5.15 [addrescape] EXAMPLE 1, 2, 3

Reviewers: svoboda

11. The value of an object with automatic storage duration is used while the object has an indeterminate representation (6.2.4, 6.7.11, 6.8).

```
void get_sign(int number, int *sign) {
  if (sign == NULL) {
    // ...
  }
  if (number > 0) {
    *sign = 1;
  } else if (number < 0) {
    *sign = -1;
  } // If number == 0, sign is not changed.
}

int is_negative(int number) {
  int sign;
  get_sign(number, &sign);
  return (sign < 0);  // Undefined Behavior, sign might not be initialized
}}
```

Cite: TS17961 5.35 [uninitref] EXAMPLE 1, 2, CERT C Rule EXP33-C 4th NCCE 4.3.9, CERT C Rec MSC22-C 3rd NCCE

Reviewers: svoboda

12. A non-value representation is read by an lvalue expression that does not have character type (6.2.6.1).

```
void f(size_t n) {
  int *a = malloc(n * sizeof(int));
  if (a != NULL) {
    for (size_t i = 0; i != n; ++i) {
      a[i] = a[i] ^ a[i]; // Undefined Behavior
    }
    // ...
    free(a);
  }
}
```

Cite: TS17961 5.35 [uninitref] EXAMPLE 3, 4

Reviewers: svoboda

13. A non-value representation is produced by a side effect that modifies any part of the object using an lvalue expression that does not have character type (6.2.6.1).

```
union {
  float f;
  int i;
} u;

u.f = 3.14;
u.i = 123;
printf("value is %f\n", u.f); // Undefined Behavior
```

Reviewers: svoboda

14. Two declarations of the same object or function specify types that are not compatible (6.2.7).

```
// In a.c:
extern int i;  // Undefined Behavior

int f(void) {
  return ++i;
}
```

```
// In b.c:
short i;        // Undefined Behavior
```

Cite: TS17961 5.13 [funcdecl] EXAMPLE 1, 2, 4

Reviewers: svoboda

15. A program requires the formation of a composite type from a variable length array type whose size is specified by an expression that is not evaluated (6.2.7).

```
void *p1(void), *p2(void);
int f(void), g(void);
int i = (f()
         ? ((int (*)[g()])p1())
         : ((int (*)[])p2()))[0][0];
// Undefined Behavior, g() may not be evaluated if f() returns 0
```

Reviewers: svoboda

16. Conversion to or from an integer type produces a value outside the range that can be represented (6.3.1.4).

```
void func(float f_a) {
  int i_a;
  i_a = f_a;  // Undefined Behavior if the integral part of f_a cannot be
represented.
}
```

Cite: CERT C Rule FLP34-C 1st NCCE 6.3.1, FLP36-C 1st NCCE 6.4.1

Reviewers: svoboda

17. Demotion of one real floating type to another produces a value outside the range that can be represented (6.3.2.1).

```
void func(double d_a, long double big_d) {
  double d_b = (float)big_d;  // Undefined Behavior, if outside range
  float f_a = (float)d_a;     // Undefined Behavior, if outside range
  float f_b = (float)big_d;   // Undefined Behavior, if outside range
}
```

Cite: CERT C Rule FLP34-C 2nd NCCE 6.3.3

Reviewers: svoboda

18. An lvalue does not designate an object when evaluated (6.3.2.1).

```
void func() {
  int *p;
  {
    int i = 1;
```

```
    p = &i;
    *p = 2;  // ok
  }
  *p = 3;    // Undefined Behavior

  int a[10];
  a[1] = 4;  // ok
  a[10] = 5; // Undefined Behavavior
}
```

Reviewers: svoboda, UBSG

19. A non-array lvalue with an incomplete type is used in a context that requires the value of the designated object (6.3.2.1).

```
struct f *p;
void g(void) {
  *p;   // Undefined Behavior
}
```

Reviewers: uecker, j.myers

20. An lvalue designating an object of automatic storage duration that could have been declared with the register storage class is used in a context that requires the value of the designated object, but the object is uninitialized. (6.3.2.1).

```
void f(void) {
  /* register */ int x;  // address of x not taken, so x could be stored in a
register
  int y = x;             // Undefined Behavior
}
```

Reviewers: svoboda, uecker, j.myers

21. An lvalue having array type is converted to a pointer to the initial element of the array, and the array object has register storage class (6.3.2.1).

```
void f(void) {
  register int a[3];
  int *p = a;  // Undefined Behavior
  a[0] = 1;
  p[0];
}
```

Reviewers: svoboda

22. An attempt is made to use the value of a void expression, or an implicit or explicit conversion (except to void) is applied to a void expression (6.3.2.2).

```
void f(int x) {
  printf("x is %d\n", x);
}

void *p = (void *) f;
int (*g)(int) = p;
int y = g(123);   // Undefined Behavior
printf("y is %d\n", y);
```

Reviewers: svoboda

23. Conversion of a pointer to an integer type produces a value outside the range that can be represented (6.3.2.3).

```
void f(void) {
  char *ptr;
  // ...
  unsigned int number = (unsigned int)ptr;  // Undefined Behavior
  // ...
}
```

Cite: TS17961 5.10 [intptrconv] EXAMPLE 1,2

Reviewers: svoboda

24. Conversion between two pointer types produces a result that is incorrectly aligned (6.3.2.3).

```
void f(void) {
  int *i_ptr;
  char c;
  i_ptr = (int *)&c;  // Undefined Behavior
  // ...
}
```

Cite: TS17961 5.11 [alignconv] EXAMPLE 1

Reviewers: svoboda

25. A pointer is used to call a function whose type is not compatible with the referenced type (6.3.2.3).

```
char *(*fp)();
void f(void) {
  char *c;
  fp = strchr;
  c = fp(12, 2);  // Undefined Behavior, incorrect arguments
}
```

Cite: TS17961 5.6 [argcomp] EXAMPLE 1

Reviewers: svoboda

26. An unmatched ' or " character is encountered on a logical source line during tokenization (6.4).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows string literals to span multiple lines:

```
char s[] = "foo
bar";   // Undefined Behavior
```

Reviewers: svoboda, UBSG

27. A reserved keyword token is used in translation phase 7 or 8 (5.1.1.2) for some purpose other than as a keyword (6.4.1).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows keywords to be used as identifiers:

```
int if = 3;  // Undefined Behavior
```

Reviewers: svoboda, UBSG

28. A universal character name in an identifier does not designate a character whose encoding falls into one of the specified ranges (6.4.2.1).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows the division sign ÷ to be used as identifiers:

```
// In UTF-8: the division sign == '÷' == U+00F7
double ONE\u00F7TWO = 0.5;  // ONE ÷ TWO, Undefined Behavior
```

Reviewers: svoboda, UBSG

29. The initial character of an identifier is a universal character name designating a digit (6.4.2.1).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that accepts universal character names that designate digits to be used as identifiers:

```
int \u0031N = 1;  // "1N", Undefined Behavior
```

Reviewers: svoboda, UBSG

30. Two identifiers differ only in nonsignificant characters (6.4.2.1).

```
// In bash/bashline.h:
extern char *bash_groupname_completion_function(const char *, int);
// Undefined Behavior, the identifier exceeds 31 characters
```

```c
// In a.c:
#include <bashline.h>

void w(const char *s, int i) {
  bash_groupname_completion_function(s, i);
  // This function was taken from GNU Bash, version 3.2.
  // https://www.gnu.org/software/bash/
}


// In b.c:
int bash_groupname_completion_funct;
// Undefined Behavior, identifier not unique within 31 characters
```

Cite: TS17961 5.13 [funcdecl] EXAMPLE 4

Reviewers: svoboda

31. The identifier __func__ is explicitly declared (6.4.2.2).

```c
void __func__(void); // Undefined Behavior
```

Reviewers: svoboda, j.myers

32. The program attempts to modify a string literal (6.4.5).

```c
void f1(void) {
  char *p = "string literal";
  p[0] = 'S';  // Undefined Behavior
  // ...
}
```

Cite: TS17961 5.28 [strmod] EXAMPLE 1, 2, 3, 4, 5

Reviewers: svoboda

33. The characters ', \, ", //, or /* occur in the sequence between the < and > delimiters, or the characters ', \, //, or /* occur in the sequence between the " delimiters, in a header name preprocessing token (6.4.7).

```c
#include "dave's_hello.h"
// Undefined Behavior
```

Reviewers: svoboda, j.myers

34. A side effect on a scalar object is unsequenced relative to either a different side effect on the same scalar object or a value computation using the value of the same scalar object (6.5).

```
#define CUBE(X) ((X) * (X) * (X))

void func(void) {
  int i = 2;
  int a = 81 / CUBE(++i);  // Undefined Behavior
  // ...
}
```

Cite: CERT C Rec PRE00-C 1st NCCE, 3rd NCCE

Reviewers: svoboda

35. An exceptional condition occurs during the evaluation of an expression (6.5.).

```
int add(void) {
  int x;
  // Initialize x with an untrusted value, which could be INT_MAX
  return x + 1;    // Undefined Behavior
}
```

Cite: TS17961 5.30 [intoflow] EXAMPLE 1

Reviewers: svoboda

36. An object has its stored value accessed other than by an lvalue of an allowable type (6.5.1).

```
void f(void) {
  if (sizeof(int) == sizeof(float)) {
    float f = 0.0f;
    int *ip = (int *)&f;
    printf("float is %f\n", f);
    (*ip)++; // Undefined Behavior
    printf("float is %f\n", f);
  }
}
```

Cite: TS17961 5.1 [ptrcomp] EXAMPLE

Reviewers: svoboda

37. A function is defined with a type that is not compatible with the type (of the expression) pointed to by the expression that denotes the called function (6.5.3.3).

```
// In somefile.c:
long f(long x) {
  return x < 0 ? -x : x;
}

// In otherfile.c:
int g(int x) {
  return f(x);  // Undefined Behavior
}
```

Cite: TS17961 5.6 [argcomp] EXAMPLE 4, 5.13 [funcdecl] EXAMPLE 3

Reviewers: svoboda

38. A member of an atomic structure or union is accessed (6.5.3.4).

```
_Atomic struct {
  int x;
} foo;
foo.x;   // Undefined Behavior
```

Reviewers: uecker, svoboda, j.myers

39. The operand of the unary * operator has an invalid value (6.5.4.2).

```
char *p = NULL;
*p;   // Undefined Behavior
```

Reviewers: svoboda, j.myers

40. A pointer is converted to other than an integer or pointer type (6.5.5).

```
struct f { struct f *x; } *p;
void g(void) {
  (struct f)p;   // Undefined Behavior
}
```

Reviewers: svoboda

41. The value of the second operand of the / or % operator is zero (6.5.6).

```
int divide(int x) {
  int y;
  // Initialize y with an untrusted value, which could be 0
  return x / y;  // Undefined Behavior
}
```

Cite: TS17961 5.26 [diverr] EXAMPLE 1

Reviewers: svoboda

42. If the quotient a/b is not representable, the behavior of both a/b and a%b (6.5.6).

```
int remainder(int x) {
  int y;
  // Initialize y with an untrusted value, which could be 0
  return x % y;  // Undefined Behavior
}
```

Cite: TS17961 5.26 [diverr] EXAMPLE 2

Reviewers: svoboda

43. Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that does not point into, or just beyond, the same array object (6.5.7).

```
#define TABLESIZE 100
static int table[TABLESIZE];

int *f(int index) {
  if (index < TABLESIZE) {
    return table + index;   // Undefined Behavior
  }
  return NULL;
}
```

Cite: TS17961 5.22 [invptr] EXAMPLE 1

Reviewers: svoboda

44. Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that points just beyond the array object and is used as the operand of a unary * operator that is evaluated (6.5.7).

```
#define MAX_MACHINE_NAME_LENGTH 64
char *get_machine_name(const char *path) {
  char *machine_name = malloc(MAX_MACHINE_NAME_LENGTH + 1);
  if (machine_name == NULL) {
    return NULL;
  }

  while (*path != '\\') {
    *machine_name++ = *path++;  // Undefined Behavior, if \ not in path
  }

  *machine_name = '\0';
```

```
  return machine_name;
}
```

Cite: TS17961 5.22 [invptr] EXAMPLE 4, 6, 10, 12

Reviewers: svoboda

45. Pointers that do not point into, or just beyond, the same array object are subtracted (6.5.7).

```
#define SIZE 256

void f(void) {
  int nums[SIZE];
  char *c_str[SIZE];
  int *next_num_ptr = nums;
  int free_bytes;

  // Increment next_num_ptr as array fills...

  free_bytes = c_str - (char **)next_num_ptr;   // Undefined Behavior
  // next_num_ptr part of name array, even if it equals c_str!
  // ...
}
```

Cite: TS17961 5.36 [ptrobj] EXAMPLE, CERT C Rule ARR36-C 1st NCCE 7.3.1

Reviewers: svoboda

46. An array subscript is out of range, even if an object is apparently accessible with the given subscript (as in the lvalue expression a[1][7] given the declaration int a[4][5]) (6.5.7).

```
enum { COLS = 5, ROWS = 7 };
static int matrix[ROWS][COLS];

void init_matrix(int x) {
  for (size_t i = 0; i != COLS; ++i) {
    for (size_t j = 0; j != ROWS; ++j) {
      matrix[i][j] = x;  // Undefined Behavior, i and j swapped
    }
  }
}
```

Cite: TS17961 5.22 [invptr] EXAMPLE 8

Reviewers: svoboda

47. The result of subtracting two pointers is not representable in an object of type ptrdiff_t (6.5.7).

```
#include <stdlib.h>

size_t size = 1 + (SIZE_MAX / 2);     // Assumes sizeof(size_t) ==
sizeof(ptrdiff_t)
char *x = malloc(size);
assert(x != NULL);
char *start = x;
char *too_far = x + size;
ptrdiff_t too_big = too_far - start;  // Undefined Behavior
```

Reviewers: svoboda

48. An expression is shifted by a negative number or by an amount greater than or equal to the width of the promoted expression (6.5.8).

```
void func(unsigned int ui_a, unsigned int ui_b) {
  unsigned int uresult = ui_a << ui_b;
  // Undefined Behavior if !( 0 < ui_b < sizeof(ui_a) / CHAR_BIT)
}
```

Cite: CERT C Rule INT34-C 1st NCCE 5.5.1, 2nd NCCE 5.5.3, 3rd NCCE 5.5.5

Reviewers: svoboda

49. An expression having signed promoted type is left-shifted and either the value of the expression is negative or the result of shifting would not be representable in the promoted type (6.5.8).

```
#include <limits.h>
#include <stddef.h>
#include <inttypes.h>

void func(signed long si_a, signed long si_b) {
  signed long result;
  if (si_a > (LONG_MAX >> si_b)) {
    // Handle Error
  } else {
    result = si_a << si_b;  // Undefined Behavior
  }
}
```

Cite: CERT C Rule INT32-C 6th NCCE 5.3.8.1, CERT C Rule INT34-C 2nd NCCE 5.5.3

Reviewers: svoboda

50. Pointers that do not point to the same aggregate or union (nor just beyond the same array object) are compared using relational operators (6.5.9).

```
struct {
  int x;
  int y;
} a, b;
if (&a.y < &b.y) {  // Undefined Behavior
  // ...
}
```

Reviewers: uecker, svoboda, j.myers

51. An object is assigned to an inexactly overlapping object or to an exactly overlapping object with incompatible type (6.5.17.2).

```
const size_t limit = sizeof(int) + 1;
char bytes[limit];
int *p1 = (int *) &(bytes[0]);
int *p2 = (int *) &(bytes[1]); // overlaps with p1 (unless sizeof(int) == 1)
*p1 = 123;
*p2 = *p1;                     // Undefined Behavior
```

Reviewers: coates, svoboda

52. An expression that is required to be an integer constant expression does not have an integer type; has operands that are not integer constants, named constants, compound literal constants, enumeration constants, character constants, predefined constants, sizeof expressions whose results are integer constants, alignof expressions, or immediately-cast floating constants; or contains casts (outside operands to sizeof and alignof operators) other than conversions of arithmetic types to integer types (6.6).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that accepts floating-point constants to be used in constant integer expressions.

```
enum people {
  Tom=1.0,           // Undefined Behavior
  Dick=2,
  Harry=3
};

struct s {
  int bit:1;
  int two_bits:2.5;  // Undefined Behavior
  int all_the_bits;
};
```

Reviewers: svoboda

53. A constant expression in an initializer is not, or does not evaluate to, one of the following: a named constant, a compound literal constant, an arithmetic constant expression, a null pointer constant, an address constant, or an address constant for a complete object type plus or minus an integer constant expression (6.6).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows function calls to be used in constant integer expressions.

```
int square(int x) {
  return x*x;
}

enum people {
  Tom=1,
  Dick=2,
  Harry=square(2)           // Undefined Behavior
};

struct s {
  int bit:1;
  int two_bits:square(2);   // Undefined Behavior
  int all_the_bits;
};
```

Reviewers: svoboda

54. An arithmetic constant expression does not have arithmetic type; has operands that are not integer constants, floating constants, named and compound literal constants of arithmetic type, character constants, predefined constants, sizeof expressions whose results are integer constants, or alignof expressions; or contains casts (outside operands to sizeof or alignof operators) other than conversions of arithmetic types to arithmetic types (6.6).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows string literals to be used in constant arithmetic expressions.

```
float f = "Hello, world!";   // Undefined Behavior
```

Reviewers: svoboda

55. The value of an object is accessed by an array-subscript [], member-access . or ->, address &, or indirection * operator or a pointer cast in creating an address constant (6.6).

```
const int primes[] = {2, 3, 5, 7};
const int *first_odd_prime = &(primes[1]); // Undefined Behavior
```

Reviewers: svoboda

56. An identifier for an object is declared with no linkage and the type of the object is incomplete after its declarator, or after its init-declarator if it has an initializer (6.7).

TODO

Reviewers:

Note: Removed from J.2. by N3244

57. A function is declared at block scope with an explicit storage-class specifier other than extern (6.7.2).

TODO

Reviewers:

Note: Removed from J.2. by N3244

58. A structure or union is defined without any named members (including those specified indirectly via anonymous structures and unions) (6.7.3.2).

```
struct {
  int:3;    // Un-named bit-field
} myStruct; // Undefined Behavior
```

Reviewers: coates, svoboda

59. An attempt is made to access, or generate a pointer to just past, a flexible array member of a structure when the referenced object provides no elements for that array (6.7.3.2).

```
#include <stdlib.h>

struct S {
  size_t len;
  char buf[];  // Flexible array member
};

const char *find(const struct S *s, int c) {
  const char *first = s->buf;
```

```
  const char *last = s->buf + s->len;

  while (first++ != last) {   // Undefined Behavior
    if (*first == (unsigned char)c) {
      return first;
    }
  }
  return NULL;
}

void g(void) {
  struct S *s = malloc(sizeof(struct S));
  if (s == NULL) {
    // Handle Error
  }
  s->len = 0;
  find(s, 'a');
}
```

Cite: CERT C Rule ARR30-C 5th NCCE 7.1.10

Reviewers: svoboda

60. When the complete type is needed, an incomplete structure or union type is not completed in the same scope by another declaration of the tag that defines the content (6.7.3.4).

TODO

Reviewers:

Note: Removed from J.2. by N3244

61. An attempt is made to modify an object defined with a const-qualified type through use of an lvalue with non-const-qualified type (6.7).

```
const int **ipp;
int *ip;
const int i = 42;

void func(void) {
  ipp = &ip;  // Constraint violation
  *ipp = &i;  // Valid
  *ip = 0;    // Undefined Behavior, modifies constant i (was 42)
}
```

Cite: CERT C Rule EXP40-C 1st NCCE 4.9.1

Reviewers: svoboda

62. An attempt is made to refer to an object defined with a volatile-qualified type through use of an lvalue with non-volatile-qualified type (6.7.4).

```c
#include <stdio.h>

void func(void) {
  static volatile int **ipp;
  static int *ip;
  static volatile int i = 0;

  printf("i = %d.\n", i);

  ipp = &ip;          // Undefined Behavior
  ipp = (int **) &ip; // Undefined Behavior
  *ipp = &i;          // Valid
  if (*ip != 0) {     // Valid
    // ...
  }
}
```

Cite: CERT C Rule EXP32-C 1st NCCE 4.2.1

Reviewers: svoboda

63. The specification of a function type includes any type qualifiers (6.7.4).

```c
typedef int fun_t(int);
const fun_t f;  // Undefined Behavior
```

Reviewers: uecker, svoboda, j.myers

64. Two qualified types that are required to be compatible do not have the identically qualified version of a compatible type (6.7.4).

```c
const struct s { int mem; } cs = { 1 };
struct s ncs; // the object ncs is modifiable
typedef int A[2][3];
const A a = {{4, 5, 6}, {7, 8, 9}}; // array of array of const int
int *pi;
const int *pci;
ncs = cs;      // valid
cs = ncs;      // Undefined Behavior: violates modifiable lvalue constraint
for =
pi = &ncs.mem; // valid

pi = &cs.mem;  // Undefined Behavior: violates type constraints for =
pci = &cs.mem; // valid
pi = a[0];     // Undefined Behavior: a[0] has type "const int *"
```

Reviewers: svoboda

65. An object which has been modified is accessed through a restrict-qualified pointer to a const-qualified type, or through a restrict-qualified pointer and another pointer that are not both based on the same object (6.7.4.2).

```
void abcabc(void) {
  char c_str[]= "abc123edf";
  char *ptr1 = c_str;
  char *ptr2 = c_str + strlen("abc");
  memcpy(ptr2, ptr1, 6);   // Undefined Behavior, objects of ptr1 & ptr2
overlap
  puts(c_str);
}
```

Cite: TS17961 5.33 [restrict] EXAMPLE 1, 2, CERT C Rule EXP43-C 2st NCCE 4.11.2.1, 3rd NCCE 4.11.2.3, 5th NCCE 4.11.4.1

Reviewers: svoboda

66. A restrict-qualified pointer is assigned a value based on another restricted pointer whose associated block neither began execution before the block associated with this pointer, nor ended before the assignment (6.7.4.2).

```
int *restrict a;
int *restrict b;
extern int c[];

int main(void) {
  c[0] = 17;
  c[1] = 18;
  a = &c[0];
  b = &c[1];
  a = b;      // Undefined Behavior
  // ...
}
```

Cite: CERT C Rule EXP43-C 1st NCCE 4.11.1.1, 4th NCCE 4.11.3.1, 6th NCCE 4.11.5.1

Reviewers: svoboda

67. A function with external linkage is declared with an inline function specifier, but is not also defined in the same translation unit (6.7.5).

```
extern inline int foo(int x);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

68. A function declared with a _Noreturn function specifier returns to its caller (6.7.5).

```c
#include <stdnoreturn.h>

_Noreturn void f(void) {
  return;  // Undefined Behavior at run-time
}
```

Reviewers: uecker, svoboda, j.myers

69. The definition of an object has an alignment specifier and another declaration of that object has a different alignment specifier (6.7.6).

TODO

Reviewers:

Note: Removed from J.2. by N3244

70. Declarations of an object in different translation units have different alignment specifiers (6.7.6).

```c
// In file1.c:
alignas(16) int a;


// In file2.c:
alignas(32) int a; // Undefined Behavior
```

Reviewers: svoboda

71. Two pointer types that are required to be compatible are not identically qualified, or are not pointers to compatible types (6.7.7.2).

```c
int i = 1, j = 2;
const int *cp = &i; // *cp is constant
int *ncp = &j;      // *ncp is modifiable
ncp = cp;           // valid
cp = ncp;           // Undefined Behavior: violates modifiable lvalue
constraint for =
```

Reviewers: svoboda

72. The size expression in an array declaration is not a constant expression and evaluates at program execution time to a nonpositive value (6.7.7.3).

```c
int size = -4;

// This creates a VLA.
int arr[size];
```

```
printf("%d\n",sizeof(arr));  // Undefined Behavior
```

Reviewers: svoboda

73. In a context requiring two array types to be compatible, they do not have compatible element types, or their size specifiers evaluate to unequal values (6.7.7.3).

```
enum { ROWS = 10, COLS = 15 };
void func(void) {
  int a[ROWS][COLS];
  int (*b)[ROWS] = a;  // Undefined Behavior
}
```

Cite: CERT C Rule EXP39-C 4th NCCE 4.8.7

Reviewers: svoboda

74. A declaration of an array parameter includes the keyword static within the [and] and the corresponding argument does not provide access to the first element of an array with at least the specified number of elements (6.7.7.4).

```
const int size = 5;

int average(int numbers[static size]) {
  int sum = 0;
  for (int i = 0; i < size; i++) {
    sum += numbers[i];
  }
  return sum / size;
}

int main(void) {
  int a[3] = { 4, 1002, 27 };
  int result = average(a);   // Undefined Behavior, array too small
  printf("Average is %d\n", result);
}
```

Reviewers: svoboda

75. A storage-class specifier or type qualifier modifies the keyword void as a function parameter type list (6.7.74).

```
void f(const void);
void g(register void);  // Undefined Behavior
```

Reviewers: svoboda

76. In a context requiring two function types to be compatible, they do not have compatible return types, or their parameters disagree in use of the ellipsis terminator or the number and type of parameters (after default argument promotion, when there is no parameter type list) (6.7.7.4).

```c
void fi(int *a) {
  (*a)++;
}

void fl(long *a) {
  (*a)++;
}

int repeat(void f(), int *a) {
  f(a);
  f(a);
  f(a);
}

int main () {
  int x = 1;
  repeat(fi, &x); // Undefined Behavior
  return 0;
}
```

Reviewers: svoboda

77. A declaration for which a type is inferred contains a pointer, array, or function declarators (6.7.10).

```c
double double_double(double x) {
  return x * 2;
}

auto (*fn)(double x) = double_double; // Undefined Behavior
```

Reviewers: svoboda

78. A declaration for which a type is inferred contains no or more than one declarators (6.7.10).

```c
auto i = 3, j = 4.5; // Undefined Behavior
```

Reviewers: svoboda

79. The value of an unnamed member of a structure or union is used (6.7.11).

TODO

Reviewers:

Note: Removed from J.2. by N3245

80. The initializer for a scalar is neither a single expression, an empty initializer, nor a single expression enclosed in braces (6.7.11).

TODO

Reviewers:

Note: Removed from J.2. by N3246

81. The initializer for a structure or union object is neither an initializer list nor a single expression that has compatible structure or union type (6.7.11).

TODO

Reviewers:

Note: Removed from J.2. by N3246

82. The initializer for an aggregate or union, other than an array initialized by a string literal, is not a brace-enclosed list of initializers for its elements or members (6.7.11).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows integer literals to be used in aggregate initializer expressions.

```
struct st {
  int value;
};

struct st s = 0;   // Undefined Behavior
```

Reviewers: svoboda

83. A function definition that does not have the asserted property is called by a function declaration or a function pointer with a type that has the unsequenced or reproducible attribute (6.7.13.8).

```
int next(int *ip) [[reproducible]] {
  return *ip++; // Undefined Behavior, not idempotent
}
```

Reviewers: svoboda

84. An identifier with external linkage is used, but in the program there does not exist exactly one external definition for the identifier, or the identifier is not used and there exist multiple external definitions for the identifier (6.9).

```
extern int x;

x = 3;   // Undefined Behavior if x is never defined
printf("x is %d!\n", x);
```

Reviewers: svoboda

85. A function that accepts a variable number of arguments is defined without a parameter type list that ends with the ellipsis notation (6.9.2).

```
// In file1.c:
int add(int first, int second) {
  return first + second;
}


// In file2.c:
#include <stdio.h>

int add(int first, int second, ...);

int main () {
  int result = add(2, 3, 5, 7, 11, -1);   // Undefined Behavior
  printf("Sum is %d\n", result);
  return 0;
}
```

Reviewers: svoboda

86. The } that terminates a function is reached, and the value of the function call is used by the caller (6.9.2).

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

int checkpass(const char *password) {
  if (strcmp(password, "pass") == 0) {
    return 1;
  }
  // Undefined Behavior, no return value
}

void func(const char *userinput) {
  if (checkpass(userinput)) {
    printf("Success\n");
```

```
  }
}
```

Cite: CERT C Rule MSC37-C 1st NCCE 15.4.1, 2nd NCCE 15.4.3, 3rd NCCE 15.4.3.1

Reviewers: svoboda

87. An identifier for an object with internal linkage and an incomplete type is declared with a tentative definition (6.9.3).

TODO

Reviewers:

Note: Removed from J.2. by N3347

88. A non-directive preprocessing directive is executed (6.10).

```
# 1234
// Undefined Behavior
```

Reviewers: svoboda

89. The token defined is generated during the expansion of a #if or #elif preprocessing directive, or the use of the defined unary operator does not match one of the two specified forms prior to macro replacement (6.10.2).

```
#define FOO 1

#define concat(uc1, uc2) uc1##uc2

#if concat(def, ined) FOO
// Undefined Behavior
```

Reviewers: svoboda

90. The #include preprocessing directive that results after expansion does not match one of the two header name forms (6.10.3).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows token concatenation in #include directives.

```
#define str(s) # s
#define xstr(s) str(s)
#define INCFILE(n) z ## n

#include xstr(INCFILE(3).h
// Undefined Behavior: #include "z3.h"
```

Reviewers: svoboda

91. The character sequence in an #include preprocessing directive does not start with a letter (6.10.3).

```
#include "2file.h"
// Undefined Behavior
```

Reviewers: svoboda

92. There are sequences of preprocessing tokens within the list of macro arguments that would otherwise act as preprocessing directives (6.10.5).

```
#include <string.h>

void func(const char *src) {
  // Validate the source string; calculate size
  char *dest;
  // malloc() destination string
  memcpy(dest, src,
    #ifdef PLATFORM1
      12  // Undefined Behavior, if memcpy() defined as macro
    #else
      24
    #endif
  );
  // ...
);
```

Cite: CERT C Rule PRE32-C 1st NCCE 2.3.1

Reviewers: svoboda

93. The result of the preprocessing operator # is not a valid character string literal (6.10.5.2).

HYPOTHETICAL COMPILABLE EXAMPLE?

```
#define s(x) #x
char *x = "s(\)";  // Ill-formed, lone single quote
```

Reviewers: svoboda, UBSG

94. The result of the preprocessing operator ## is not a valid preprocessing token (6.10.5.3).

HYPOTHETICAL COMPILABLE EXAMPLE?

```
#define my_join(a, b) a ## b
char q[] = my_join(!, !);   // Ill-formed
```

Reviewers: svoboda

95. The #line preprocessing directive that results after expansion does not match one of the two well-defined forms, or its digit sequence specifies zero or a number greater than 2147483647 (6.10.6).

```
#line 10000000000
// Undefined Behavior, > 2^32
```

Reviewers: svoboda

96. A non-STDC #pragma preprocessing directive that is documented as causing translation failure or some other form of undefined behavior is encountered (6.10.8).

```
#pragma options align=foobar
// Undefined Behavior with GCC on Darwin platforms (e.g. Mac). See
// https://gcc.gnu.org/onlinedocs/gcc/Darwin-Pragmas.html
```

Reviewers: svoboda

97. A #pragma STDC preprocessing directive does not match one of the well-defined forms (6.10.8).

```
#pragma STDC FP_CONTRACT FOOBAR
// Undefined Behavior
```

Reviewers: svoboda

98. The name of a predefined macro, or the identifier defined, is the subject of a #define or #undef preprocessing directive (6.10.10).

```
#define __FILE__ source.c
// Undefined Behavior
```

Reviewers: svoboda

99. An attempt is made to copy an object to an overlapping object by use of a library function, other than as explicitly allowed (e.g., memmove) (Clause 7).

```
#include <string.h>

void func(void) {
  char c_str[]= "test string";
  char *ptr1 = c_str;
  char *ptr2;

  ptr2 = ptr1 + 3;
  memcpy(ptr2, ptr1, 6);  // Undefined Behavior because of overlapping
objects
}
```

Cite: CERT C Rule EXP43-C 4th NCCE 4.11.3.1

Reviewers: svoboda

100. A file with the same name as one of the standard headers, not provided as part of the implementation, is placed in any of the standard places that are searched for included source files (7.1.2).

```
#include "stdio.h"
// Undefined Behavior, distinct from <stdio.h> */
```

Cite: CERT C Rec PRE04-C 1st NCCE

Reviewers: svoboda

101. A header is included within an external declaration or definition (7.1.2).

```
struct st {
  int s;
#include <stdio.h>
// Undefined Behavior
};
```

Reviewers: svoboda

102. A function, object, type, or macro that is specified as being declared or defined by some standard header is used before any header that declares or defines it is included (7.1.2).

```
#include <stdio.h>

double sin(double x) {
  return x - 3.0;
}

#include <math.h>
// Undefined Behavior, sin() already defined

int main(void) {
  double p = 3.14; // almost pi
  printf("sin( %f) is %f\n", p, sin(p));
  return 0;
}
```

Reviewers: svoboda

103. A standard header is included while a macro is defined with the same name as a keyword (7.1.2).

```
#define do int x;

#include <stdio.h>
// Undefined Behavior, sin() already defined
```

Reviewers: svoboda

104. The program attempts to declare a library function itself, rather than via a standard header, but the declaration does not have external linkage (7.1.2).

```
#include <stddef.h>

static void *malloc(size_t nbytes); // Undefined Behavior
```

Cite: CERT C Rule DCL37-C 4th NCCE 3.4.7

Reviewers: svoboda, chrisbazley

105. The program declares or defines a reserved identifier, other than as allowed by 7.1.4 (7.1.3).

```
// Identifiers that begin with _ are reserved at file scope.
static const unsigned int _max_limit = 1024; // Undefined Behavior
unsigned int _limit = 100; // Undefined Behavior
struct _data; // Undefined Behavior

/* Identifiers that begin with __ or _ followed by an uppercase letter
   are reserved for any use. */
unsigned int getValue(unsigned int __count) { // Undefined Behavior
  static unsigned int _CallCount; // Undefined Behavior
  _CallCount++;
  return __count < _limit ? __count : _limit;
}
```

Cite: TS17961 5.44 [resident] EXAMPLE 1, 2, 4, 6, 8

Reviewers: svoboda, chrisbazley

106. The program removes the definition of a macro whose name begins with an underscore and either an uppercase letter or another underscore (7.1.3).

```
#undef __STDC_UTF_32__
// Undefined Behavior
```

Reviewers: svoboda, j.myers

107. An argument to a library function has an invalid value or a type not expected by a function with a variable number of arguments (7.1.4).

```
#include <stdio.h>

printf("Hello, %s!\n", 123);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

108. The pointer passed to a library function array parameter does not have a value such that all address computations and object accesses are valid (7.1.4).

```
void f1(size_t nchars) {
  char *p = malloc(nchars);
  const size_t n = nchars + 1;
  if (p) {
    memset(p, 0, n); // Undefined Behavior, 1-byte buffer overflow
    // ...
  }
}
```

Cite: TS17961 5.20 [libptr] EXAMPLE 1, 2, 3, 4, 5.22 [invptr] EXAMPLE 13, 5.31 [nonnullcs] EXAMPLE 2, 5.37 [taintstrcpy] EXAMPLE, 5.40 [taintformatio] EXAMPLE 2

Reviewers: svoboda

109. The macro definition of assert is suppressed to access an actual function (7.2).

```
#include <assert.h>

typedef void (*handler_type)(int);

void execute_handler(handler_type handler, int value) {
  handler(value);
}

void func(int e) {
  execute_handler(&(assert), e < 0);  // Undefined Behavior
}
```

Cite: CERT C Rule MSC38-C 1st NCCE 15.5.1

Reviewers: svoboda

110. The argument to the assert macro does not have a scalar type (7.2).

```
#include <assert.h>

int a[5];   // An array is considered an aggregate type (C23 s6.2.5p26).
assert(a);  // Undefined Behavior
```

Reviewers: svoboda

111. The CX_LIMITED_RANGE, FENV_ACCESS, or FP_CONTRACT pragma is used in any context other than outside all external declarations or preceding all explicit declarations and statements inside a compound statement (7.3.4, 7.6.1, 7.12.2).

```
void f(void) {
  float a = 4.0 / 7;
  #pragma STDC FP_CONTRACT OFF
  a *= 7;   // Undefined Behavior
  printf("A is %f\n", a);
}
```

Reviewers: svoboda

112. The value of an argument to a character handling function is neither equal to the value of EOF nor representable as an unsigned char (7.4).

```
size_t count_preceding_whitespace(const char *s) {
  const char *t = s;
  size_t length = strlen(s) + 1;

  while (isspace(*t) && (t - s < length)) {  // Undefined Behavior, if char
is signed
    ++t;
  }
  return t - s;
}
```

Cite: TS17961 5.32 [chrsgnext] EXAMPLE

Reviewers: svoboda

113. A macro definition of errno is suppressed to access an actual object, or the program defines an identifier with the name errno (7.5).

```
extern int errno;  // Undefined Behavior
```

Cite: TS17961 5.44 [resident] EXAMPLE 1, CERT C Rule MSC38-C 2nd NCCE 15.5.3

Reviewers: svoboda

114. Part of the program tests floating-point status flags, sets floating-point control modes, or runs under non-default mode settings, but was translated with the state for the FENV_ACCESS pragma "off" (7.6.1).

```
#pragma STDC FENV_ACCESS OFF

int set_excepts;
feclearexcept(FE_INVALID);
set_excepts = fetestexcept(FE_INVALID); // Undefined Behavior
```

```c
if (set_excepts & FE_INVALID) {
  puts("INVALID");
}
```

Reviewers: svoboda

115. The exception-mask argument for one of the functions that provide access to the floating-point status flags has a nonzero value not obtained by bitwise OR of the floating-point exception macros (7.6.4).

```c
#pragma STDC FENV_ACCESS ON
int set_excepts = fetestexcept(1); // Undefined Behavior
```

Reviewers: svoboda

116. The fesetexceptflag function is used to set floating-point status flags that were not specified in the call to the fegetexceptflag function that provided the value of the corresponding fexcept_t object (7.6.4.5).

```c
#pragma STDC FENV_ACCESS ON

fexcept_t excepts;
fesetexceptflag(&excepts,FE_ALL_EXCEPT);
// Undefined Behavior, excepts not set by fegetexceptflag
```

Reviewers: svoboda

117. The argument to fesetenv or feupdateenv is neither an object set by a call to fegetenv or feholdexcept, nor is it an environment macro (7.6.6.3, 7.6.6.4).

```c
#pragma STDC FENV_ACCESS ON

fenv_t fenv;
fesetenv(&fenv);
// Undefined Behavior, fenv not initialized
```

Reviewers: svoboda

118. The value of the result of an integer arithmetic or conversion function cannot be represented (7.8.2.1, 7.8.2.2, 7.8.2.3, 7.8.2.4, 7.24.6.1, 7.24.6.2, 7.24.1).

```c
intmax_t x = INTMAX_MIN;
intmax_t px = imaxabs(x);  // Undefined Behavior
```

Reviewers: svoboda

119. The program modifies the string pointed to by the value returned by the setlocale function (7.11.1.1).

```
void f1(void) {
  char *locale = setlocale(LC_ALL, 0);
  if (locale != NULL) {
    char *cats[8];
    char *sep = locale;
    cats[0] = locale;
    int i;

    if (sep) {
      for (i = 0; (sep = strstr(sep, ";:")) && i < 8; ++i) {
        *sep = '\0';   // Undefined Behavior
        cats[i] = ++sep;
      }
    }
  }
}
```

Cite: TS17961 5.29 [libmod] EXAMPLE 1

Reviewers: svoboda

120. A pointer returned by the setlocale function is used after a subsequent call to the function, or after the calling thread has exited (7.11.1.1).

```
#include <locale.h>

char *locale1 = setlocale(LC_ALL, "");
assert(locale1);

// ...

char *locale2 = setlocale(LC_ALL, "");
int size = strlen(locale1);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

121. The program modifies the structure pointed to by the value returned by the localeconv function (7.11.2.1).

```
void f2(void) {
  struct lconv *conv = localeconv();
  if ('\0' == conv->decimal_point[0]) {
    conv->decimal_point = ".";   // Undefined Behavior
  }
  if ('\0' == conv->thousands_sep[0]) {
    conv->thousands_sep = ",";   // Undefined Behavior
```

```
  }
}
```

Cite: TS17961 5.29 [libmod] EXAMPLE 2

Reviewers: svoboda

122. A macro definition of math_errhandling is suppressed or the program defines an identifier with the name math_errhandling (7.12).

```
#include <math.h>

int math_errhandling;  // Undefined Behavior
```

Reviewers: svoboda

123. An argument to a floating-point classification or comparison macro is not of real floating type (7.12.3, 7.12.17).

```
double complex p = CMPLX( 2, 3); // 2 + 3i
if (isfinite(p) {  // Undefined Behavior
  // ...
}
```

Reviewers: svoboda

124. A macro definition of setjmp is suppressed to access an actual function, or the program defines an external identifier with the name setjmp (7.13).

```
#include <setjmp.h>

int setjmp(char *foo);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

125. An invocation of the setjmp macro occurs other than in an allowed context (7.13.2.1).

```
jmp_buf buf;

void f(void) {
  int i = setjmp(buf);  // Undefined Behavior
  if (i == 0) {
    g();
  } else {
    // longjmp was invoked
  }
}

void g(void) {
  // ...
```

```
  longjmp(buf, 1);
}
```

Cite: CERT C Rec MSC22-C 1st NCCE

Reviewers: svoboda

126. The longjmp function is invoked to restore a nonexistent environment (7.13.2.1).

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

static jmp_buf buf;
static void bad(void);

static void g(void) {
  if (setjmp(buf) == 0) {
    printf("setjmp() invoked\n");
  } else {
    printf("longjmp() invoked\n");
  }
}

static void f(void) {
  g();
}

static void setup(void) {
  f();
}

void do_stuff(void) {
  void (*b)(void) = bad;
  // ...
  longjmp(buf, 1);   // Undefined Behavior
}

static void bad(void) {
  printf("Should not be called!\n");
  exit(1);
}

int main(void) {
  setup();
  do_stuff();
}
```

Cite: CERT C Rec MSC22-C,2nd NCCE

Reviewers: svoboda

127. After a longjmp, there is an attempt to access the value of an object of automatic storage duration that does not have volatile-qualified type, local to the function containing the invocation of the corresponding setjmp macro, that was changed between the setjmp invocation and longjmp call (7.13.2.1).

```
jmp_buf buf;

void f(void) {
  int i = 0;
  if (setjmp(buf) != 0) {
    printf("%i\n", i);      // Undefined Behavior
    // ...
  }
  i = 2;
  g();
}

void g(void) {
  // ...
  longjmp(buf, 1);
}
```

Cite: CERT C Rec MSC22-C 3rd NCCE

Reviewers: svoboda

128. The program specifies an invalid pointer to a signal handler function (7.14.1.1).

```
#include <signal.h>

void *handler = NULL;
signal(SIG_IGN, handler);  // Undefined Behavior
```

Reviewers: svoboda

129. A signal handler returns when the signal corresponded to a computational exception (7.14.1.1).

```
#include <errno.h>
#include <limits.h>
#include <signal.h>
#include <stdlib.h>

volatile sig_atomic_t denom;

void sighandle(int s) {
  // Fix the offending volatile
```

```
  if (denom == 0) {
    denom = 1;
  }
}

int main(int argc, char *argv[]) {
  if (argc < 2) {
    return 0;
  }

  char *end = NULL;
  long temp = strtol(argv[1], &end, 10);

  if (end == argv[1] || 0 != *end ||
     ((LONG_MIN == temp || LONG_MAX == temp) && errno == ERANGE)) {
    // Handle Error
  }

  denom = (sig_atomic_t)temp;
  signal(SIGFPE, sighandle);

  long result = 100 / (long)denom;  // Undefined Behavior
  return 0;
}
```

Cite: CERT C Rule: SIG35-C, 1st NCCE

Reviewers: svoboda

130. A signal handler called in response to SIGFPE, SIGILL, SIGSEGV, or any other implementation-defined value corresponding to a computational exception returns (7.14.1.1).

See UB Example #129.

Cite: CERT C Rule SIG35-C 1st NCCE 12.4.1

Reviewers: svoboda

131. A signal occurs as the result of calling the abort or raise function, and the signal handler calls the raise function (7.14.1.1).

```
void term_handler(int signum) {
  // SIGTERM handling specific ...
}

void int_handler(int signum) {
  // SIGINT handling specific ...
  if (raise(SIGTERM) != 0) {   // Undefined Behavior
```

```
    // Handle Error
  }
}

int main(void) {
  if (signal(SIGTERM, term_handler) == SIG_ERR) {
    // Handle Error
  }
  if (signal(SIGINT, int_handler) == SIG_ERR) {
    // Handle Error
  }

  // Program code ...

  if (raise(SIGINT) != 0) {
    // Handle Error
  }

  // More code ...

  return EXIT_SUCCESS;
}
```

Cite: TS17961 5.5 [asyncsig] EXAMPLE 2, CERT C Rule SIG30-C 3rd NCCE 12.1.5

Reviewers: svoboda

132. A signal occurs other than as the result of calling the abort or raise function, and the signal handler refers to an object with static or thread storage duration that is not a lock-free atomic object other than by assigning a value to an object declared as volatile sig_atomic_t, or calls any function in the standard library other than the abort function, the _Exit function, the quick_exit function, the functions in <stdatomic.h> (except where explicitly stated otherwise) when the atomic arguments are lock-free, the atomic_is_lock_free function with any atomic argument, or the signal function (for the same signal number) (7.14.1.1).

```
#define MAX_MSG_SIZE 24
char *err_msg;

void handler(int signum) {
  if ((strcpy(err_msg, "SIGINT detected.")) == err_msg){ // Undefined
Behavior
    // ...
  }
}
```

```c
int main(void) {
  signal(SIGINT, handler);

  err_msg = malloc(MAX_MSG_SIZE);
  if (err_msg == NULL) {
    /// Handle Error
  }
  if ((strcpy(err_msg, "No errors yet.")) == err_msg) {
    // ...
  }

  // Main code loop ...

  return EXIT_SUCCESS;
}
```

Cite: TS17961 5.3 [accsig] EXAMPLE, 5.5 [asyncsig] EXAMPLE 1, 3

Reviewers: svoboda

133. The value of errno is referred to after a signal occurred other than as the result of calling the abort or raise function and the corresponding signal handler obtained a SIG_ERR return from a call to the signal function (7.14.1.1).

```c
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>

typedef void (*pfv)(int);

void handler(int signum) {
  pfv old_handler = signal(signum, SIG_DFL);
  if (old_handler == SIG_ERR) {
    perror("SIGINT handler"); // Undefined Behavior
    // Handle Error
  }
}
int main(void) {
  pfv old_handler = signal(SIGINT, handler);
  if (old_handler == SIG_ERR) {
    perror("SIGINT handler");
    // Handle Error
  }

  // Main code loop

  return EXIT_SUCCESS;
}
```

Cite: CERT C Rule ERR32-C 1st NCCE 13.2.1

Reviewers: svoboda

134. A signal is generated by an asynchronous signal handler (7.14.1.1).

```c
#include <setjmp.h>
#include <signal.h>
#include <stdlib.h>

enum { MAXLINE = 1024 };
static jmp_buf env;

void handler(int signum) {
  longjmp(env, 1);
}

void log_message(char *info1, char *info2) {
  static char *buf = NULL;
  static size_t bufsize;
  char buf0[MAXLINE];

  if (buf == NULL) {
    buf = buf0;
    bufsize = sizeof(buf0);
  }

  // Try to fit a message into buf, else reallocate
  // it on the heap and then log the message.

  // Undefined Behavior if SIGINT is raised here

  if (buf == buf0) {
    buf = NULL;
  }
}

int main(void) {
  if (signal(SIGINT, handler) == SIG_ERR) {
    // Handle Error
  }
  char *info1;
  char *info2;

  // info1 and info2 are set by user input here

  if (setjmp(env) == 0) {
    while (1) {
      // Main loop program code ...
```

```
      log_message(info1, info2);
      // More program code ...
    }
  } else {
    log_message(info1, info2);
  }
  return 0;
}
```

Cite: CERT C Rule SIG30-C 2nd NCCE 12.1.3

Reviewers: svoboda

135. The signal function is used in a multi-threaded program (7.14.1.1).

```
#include <signal.h>
#include <stddef.h>
#include <threads.h>

volatile sig_atomic_t flag = 0;

void handler(int signum) {
  flag = 1;
}

// Runs until user sends SIGUSR1
int func(void *data) {
  while (!flag) {
    // ...
  }
  return 0;
}

int main(void) {
  signal(SIGUSR1, handler);  // Undefined Behavior
  thrd_t tid;

  if (thrd_success != thrd_create(&tid, func, NULL)) {
    // Handle Error
  }
  // ...
  return 0;
}
```

Cite: CERT C Rule CON37-C 1st NCCE 14.8.1

Reviewers: svoboda

136. A function with a variable number of arguments attempts to access its varying arguments other than through a properly declared and initialized va_list object, or before the va_start macro is invoked (7.16, 7.16.1.1, 7.16.1.4).

```c
#include <stdio.h>
#include <stdarg.h>

void f(int last, ...) {
  va_list args;
  int number = va_arg(args, int);  // Undefined Behavior
  va_start( args, last);           // Oops, should precede va_args!
  printf("The number is %d\n", number);
  va_end(args);
}

int main(void) {
  f(1, 2, 3);
  return 0;
}
```

Reviewers: svoboda, j.myers

137. The macro va_arg is invoked using the parameter ap that was passed to a function that invoked the macro va_arg with the same parameter (7.16).

```c
#include <stdio.h>
#include <stdarg.h>

void g(va_list args) {
  int number = va_arg(args, int);
  printf("The first variadic number is %d\n", number);
}

void f(int last, ...) {
  va_list args;
  va_start( args, last);
  g(args);
  int number = va_arg(args, int);  // Undefined Behavior
  printf("The next variadic number is %d\n", number);
  va_end(args);
}

void main(void) {
  f(1, 2, 3);
}
```

Reviewers: svoboda, j.myers

138. A macro definition of va_start, va_arg, va_copy, or va_end is suppressed to access an actual function, or the program defines an external identifier with the name va_copy or va_end (7.16.1).

```c
#include <stdarg.h>

// Undefined Behavior
#undef va_arg

int va_arg(void) {
  return 123;
}
```

Reviewers: svoboda

139. The va_start or va_copy macro is invoked without a corresponding invocation of the va_end macro in the same function, or vice versa (7.16.1, 7.16.1.2, 7.16.1.3, 7.16.1.4).

```c
#include <stdarg.h>

void f(int last, ...) {
  va_list args;
  va_start( args, last);
  // Undefined Behavior, missing va_end(args)
}
```

Reviewers: svoboda, j.myers

140. The va_arg macro is invoked when there is no actual next argument, or with a specified type that is not compatible with the promoted type of the actual next argument, with certain exceptions (7.16.1.1).

```c
enum { va_eol = -1 };

unsigned int average(int first, ...) {
  unsigned int count = 0;
  unsigned int sum = 0;
  int i = first;
  va_list args;

  va_start(args, first);

  while (i != va_eol) {
    sum += i;
    count++;
    i = va_arg(args, int);
  }
```

```
  va_end(args);
  return(count ? (sum / count) : 0);
}


int avg = average(1, 4, 6, 4, 1); // Undefined Behavior
```

Cite: CERT C Rec DCL10-C 1st NCCE, CERT C Rule MSC39-C 1st NCCE 15.6.1

Reviewers: svoboda

141. The type parameter to the va_arg macro does not name an object type (7.16.1.1).

HYPOTHETICAL COMPILABLE EXAMPLE? (ideally one that replaces a with a function pointer example)

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

int main(void) {
  int x[] = {1, 2};
  int y[] = {3, 4};
  my_printf("My data", x, y);
}

void my_printf(const char *prefix, ...) {
  va_list args;
  int *x;
  int *y;
  va_start( args, prefix);
  x = va_arg( args, int *);     // Valid
  y = va_arg( args, void());    // Undefined Behavior
  va_end( args);
  printf("%s: [%d, %d], [%d, %d]\n", prefix, x[0], x[1], y[0], y[1]);
}
```

Reviewers: svoboda, UBSG

142. Using a null pointer constant in form of an integer expression as an argument to a ... function and then interpreting it as a void * or char * (7.16.1.1).

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

int contains_zero(size_t count, va_list ap) {
  for (size_t i = 1; i < count; ++i) {
    if (va_arg(ap, double) == 0.0) {
```

```
      return 1;
    }
  }
  return 0;
}

int print_reciprocals(size_t count, ...) {
  va_list ap;
  va_start(ap, count);

  if (contains_zero(count, ap)) {  // Undefined Behavior
    va_end(ap);
    return 1;
  }

  for (size_t i = 0; i < count; ++i) {
    printf("%f ", 1.0 / va_arg(ap, double));
  }

  va_end(ap);
  return 0;
}
```

Cite: CERT C Rec DCL10-C 1st NCCE, CERT C Rule MSC39-C 1st NCCE 15.6.1

Reviewers: svoboda

143. The va_copy or va_start macro is invoked to initialize a va_list that was previously initialized by either macro without an intervening invocation of the va_end macro for the same va_list (7.16.1.2, 7.16.1.4).

```
#include <stdarg.h>

void f(int last, ...) {
  va_list args;
  va_start( args, last);
  va_start( args, last);   // Undefined Behavior
  va_end(args);
}
```

Reviewers: svoboda, j.myers

144. The va_start macro is invoked with additional arguments that include unbalanced parentheses, or unrecognized preprocessing tokens (7.16.1.4).

```
#include <stdarg.h>

#define CUBE(X) ((X) * (X) * (X))

void f(int last, ...) {
```

```
  va_list args;
  va_start( args, CUBE(last));   // Undefined Behavior
  va_end(args);
}
```

Reviewers: svoboda

145. The macro definition of a generic function is suppressed to access an actual function (7.17.1, 7.18).

```
#include <stdbit.h>

#undef stdc_has_single_bit
// Undefined Behavior
```

Reviewers: svoboda

146. The type parameter of an offsetof macro defines a new type (7.21).

HYPOTHETICAL COMPILABLE EXAMPLE?

```
int plus(int a, int b) {
  return a+b;
}

typedef int binary_f(int, int);
binary_f *add = plus;

typedef struct st {
  int num1;
  int num2;
} binary_s;

size_t z = offsetof( int (*)(int, int), num2);  // Undefined Behavior
```

NOTE: j.myers says:

This was meant to be for a structure or union defined in offsetof, and also note that the entry in C23 Annex J was a mistake (Annex J not updated to reflect the resolution to CD2 DE-137), I fixed it to reflect the correct UB in commit 486be7314de9a103179134a0c469ee3689f8b641.

Reviewers: svoboda

147. When program execution reaches an unreachable() macro invocation (7.21.1).

```
if (x == 0) {
  unreachable(); // Undefined Behavior
}
```

Reviewers: svoboda

148. Arbitrarily copying or changing the bytes of or copying from a non-null pointer into a nullptr_t object and then reading that object (7.21.2).

```
int i = 1;
int *pi = &i;
nullptr_t n = nullptr;
memcpy(&n, pi, sizeof(n));
pi = n; // Undefined Behavior
```

Reviewers: svoboda

149. The member-designator parameter of an offsetof macro is an invalid right operand of the . operator for the type parameter, or designates a bit-field (7.21).

EXTENDED COMPILABLE EXAMPLE: Consider a platform that allows invalid member-designator parameteteters in offset macros:

```
typedef struct st {
  int num1;
  int num2;
} binary_s;

size_t z = offsetof( binary_s, num3);   // Undefined Behavior, should be num2
```

Reviewers: svoboda

150. The argument in an instance of one of the integer-constant macros is not a decimal, octal, or hexadecimal constant, or it has a value that exceeds the limits for the corresponding type (7.22.4).

```
unsigned char i = UINT8_C(0x123); // Undefined Behavior, 0x123 > 2^8
```

Reviewers: svoboda

151. A byte input/output function is applied to a wide-oriented stream, or a wide character input/output function is applied to a byte-oriented stream (7.23.2).

```
#include <stdio.h>
#include <wchar.h>

int main(void) {
  FILE *in = fopen("foo.txt", "r");

  wchar_t wide_line[80];
  fgetws(wide_line, sizeof(wchar_t) * sizeof(wide_line), in);
  // The stream is now oriented for wide characters
```

```
  wprintf(L"The first line is: %ls", wide_line);

  char line[80];
  fgets(line, sizeof(line), in);  // Undefined Behavior
  printf("The second line is: %s", line);

  return 0;
}
```

Reviewers: svoboda, j.myers

152. Use is made of any portion of a file beyond the most recent wide character written to a wide-oriented stream (7.23.2).

TODO

Reviewers:

Note: Removed from J.2. by N3064

153. The value of a pointer to a FILE object is used after the associated file is closed (7.23.3).

```
#include <stdio.h>

int close_stdout(void) {
  if (fclose(stdout) == EOF) {
    return -1;
  }

  printf("stdout successfully closed.\n");  // Undefined Behavior
  return 0;
}
```

Cite: CERT C Rule FIO46-C 1st NCCE 10.12.1

Reviewers: svoboda

154. The stream for the fflush function points to an input stream or to an update stream in which the most recent operation was input (7.23.5.2).

```
#include <stdio.h>

FILE *f = fopen("foo", "r");
fflush(f);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

155. The string pointed to by the mode argument in a call to the fopen function does not exactly match one of the specified character sequences (7.23.5.3).

```c
#include <stdio.h>

FILE *f = fopen("foo", "read");  // Undefined Behavior
```

Reviewers: svoboda, j.myers

156. An output operation on an update stream is followed by an input operation without an intervening call to the fflush function or a file positioning function, or an input operation on an update stream is followed by an output operation with an intervening call to a file positioning function (7.23.5.3).

```c
void f(const char *filename, char append_data[BUFSIZ]) {
  char data[BUFSIZ];
  FILE *file;

  file = fopen(filename, "a+");
  if (file == NULL) {
    // ...
  }
  if (fwrite(append_data, sizeof(char), BUFSIZ, file) != BUFSIZ) {
    // ...
  }
  if (fread(data, sizeof(char), BUFSIZ, file) != 0) {  // Undefined Behavior
    // ...
  }

  fclose(file);
}
```

Cite: TS17961 5.27 [ioileave] EXAMPLE

Reviewers: svoboda

157. An attempt is made to use the contents of the array that was supplied in a call to the setvbuf function (7.23.5.6).

```c
#include <stdio.h>

#define SIZE 1024
char buf[SIZE];
FILE *file = fopen("foo", "r");
if (file == NULL ||
    setvbuf(file, buf, _IOFBF, SIZE) != 0) {
  // Handle Error
}
```

```
// ...
putchar(buf[0]);  // Undefined Behavior
```

Reviewers: svoboda, j.myers, chrisbazley

158. There are insufficient arguments for the format in a call to one of the formatted input/output functions, or an argument does not have an appropriate type (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

```
void f(void) {
  const char *error_msg = "Resource not available to user.";
  int error_type = 3;
  // ...
  printf("Error (type %s): %d\n", error_type, error_msg);  // Undefined
Behavior
}
```

Cite: TS17961 5.45 [invfmtstr] EXAMPLE, CERT C Rec DCL10-C 2nd NCCE

Reviewers: svoboda

159. The format in a call to one of the formatted input/output functions or to the strftime or wcsftime function is not a valid multibyte character sequence that begins and ends in its initial shift state (7.23.6.1, 7.23.6.2, 7.29.3.5, 7.31.2.1, 7.31.2.2, 7.31.5.1).

```
#include <stdio.h>
#include <locale.h>

setlocale(LC_ALL, "UTF-8");

// In UTF-8: the Euro symbol == '€' == U+20AC == \xE2 \x82 \xAC == \342 \202
\254
const char s[] = {'\xE2', '\0'};  // invalid UTF-8
printf(s);   // Undefined Behavior in UTF-8 locale
```

Reviewers: svoboda, UBSG, j.myers

160. In a call to one of the formatted output functions, a precision appears with a conversion specifier other than those described (7.23.6.1, 7.31.2.1).

```
#include <stdio.h>

void f(char c) {
  printf("%.3c", c);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

161. A conversion specification for a formatted output function uses an asterisk to denote an argument-supplied field width or precision, but the corresponding argument is not provided (7.23.6.1, 7.31.2.1).

```c
#include <stdio.h>

void f(int i) {
  printf("%*iX", i);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

162. A conversion specification for a formatted output function uses a # or 0 flag with a conversion specifier other than those described (7.23.6.1, 7.31.2.1).

```c
#include <stdio.h>

void f(char *s) {
  printf("%0s", s);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

163. A conversion specification for one of the formatted input/output functions uses a length modifier with a conversion specifier other than those described (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

```c
#include <stdio.h>

void f(int *pi) {
  printf("%lp", pi);  // 'l' not defined for %p
}
```

Reviewers: svoboda, j.myers

164. An s conversion specifier is encountered by one of the formatted output functions, and the argument is missing the null terminator (unless a precision is specified that does not require null termination) (7.23.6.1, 7.31.2.1).

```c
char str[3] = "abc";  // str not null-terminated!
printf("%s\n", str);  // Undefined Behavior
```

Cite: TS17961 5.31 [nonnullcs] EXAMPLE 1

Reviewers: svoboda

165. An n conversion specification for one of the formatted input/output functions includes any flags, an assignment-suppressing character, a field width, or a precision (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

```c
#include <stdio.h>

void f(int *pi) {
  printf("%-n", pi);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

166. A % conversion specifier is encountered by one of the formatted input/output functions, but the complete conversion specification is not exactly %% (7.23.6.1, 7.23.6.2, 7.31.2.1, 7.31.2.2).

```c
#include <stdio.h>

void f(void) {
  printf("%-%");  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

167. An invalid conversion specification is found in the format for one of the formatted input/output functions, or the strftime or wcsftime function (7.23.6.1, 7.23.6.2, 7.29.3.5, 7.31.2.1, 7.31.2.2, 7.31.5.1).

```c
#include <stdio.h>

void f(int i) {
  printf("%q", i);  // Undefined behavior, %q not defined
}
```

Reviewers: svoboda, j.myers

168. The number of characters or wide characters transmitted by a formatted output function (or written to an array, or that would have been written to an array) is greater than INT_MAX (7.23.6.1, 7.31.2.1).

```c
#include <stdio.h>

void f(int i) {
  printf("%*iX", INT_MAX, i);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

169. The number of input items assigned by a formatted input function is greater than INT_MAX (7.23.6.2, 7.31.2.2).

```
#include <stdio.h>
#include <limits.h>
#include <stdarg.h>

// Assume this function is called with >INT_MAX arguments
void f(int unused, ...) {
  va_list args;
  va_start( args, unused);

  static unsigned int size = (unsigned int) INT_MAX + 1U;

  char format_string[size*2 + 1]; // will be "%c%c%c..."
  // This assumes that SIZE_MAX > 2*UINT_MAX
  for (unsigned int i = 0; i < size; i += 2) {
    format_string[i] = '%';
    format_string[i+1] = 'c';
  }
  format_string[size*2] = '\0';

  vscanf(format_string, args);    // Undefined Behavior
}
```

Reviewers: svoboda

170. The result of a conversion by one of the formatted input functions cannot be represented in the corresponding object, or the receiving object does not have an appropriate type (7.23.6.2, 7.31.2.2).

```
long num_long;

if (scanf("%ld", &num_long) != 1) { // Undefined Behavior
  // Handle Error
}
```

Cite: CERT C Rec INT05-C 1st NCCE

Reviewers: svoboda

171. A c, s, or [conversion specifier is encountered by one of the formatted input functions, and the array pointed to by the corresponding argument is not large enough to accept the input sequence (and a null terminator if the conversion specifier is s or []) (7.23.6.2, 7.31.2.2).

```
char buf[BUF_LENGTH];
fscanf(stdin, "%s", buf);  // Undefined Behavior
```

Cite: TS17961 5.40 [taintformatio] EXAMPLE 1

Reviewers: svoboda

172. A c, s, or [conversion specifier with an l qualifier is encountered by one of the formatted input functions, but the input is not a valid multibyte character sequence that begins in the initial shift state (7.23.6.2, 7.31.2.2).

```
#include <stdio.h>
#include <locale.h>

setlocale(LC_ALL, "UTF-8");
// In UTF-8: the Euro symbol == '€' == U+20AC == \xE2 \x82 \xAC == \342 \202
\254
const char invalid[] = {'\xE2', '\0'};  // invalid UTF-8
char c;
sscanf(invalid, "%c", &c);  // Undefined Behavior in UTF-8 locale
```

Reviewers: svoboda, j.myers

173. The input item for a %p conversion by one of the formatted input functions is not a value converted earlier during the same program execution (7.23.6.2, 7.31.2.2).

```
#include <stdio.h>

char addr[] = "0x12345678"; // not produced by this code
void *ptr;
sscanf( addr, "%p", &ptr);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

174. The vfprintf, vfscanf, vprintf, vscanf, vsnprintf, vsprintf, vsscanf, vfwprintf, vfwscanf, vswprintf, vswscanf, vwprintf, or vwscanf function is called with an improperly initialized va_list argument, or the argument is used (other than in an invocation of va_end) after the function returns (7.23.6.8, 7.23.6.9, 7.23.6.10, 7.23.6.11, 7.23.6.12, 7.23.6.13, 7.23.6.14, 7.31.2.5, 7.31.2.6, 7.31.2.7, 7.31.2.8, 7.31.2.9, 7.31.2.10).

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

void f(int first, ...) {
  int local = 0;
  va_list args;
  va_start( args, local);          // Oops, not first fixed arg!
  vprintf("Hello, %s\n", args);    // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

175. The contents of the array supplied in a call to the fgets or fgetws function are used after a read error occurred (7.23.7.2, 7.31.3.2).

```
const int buf_size = 100;
char buf[buf_size];
FILE *f = fopen("foo", "r");
if (f == NULL) {
  printf("Can't open foo\n");
}
fgets( buf, buf_size, f);       // Oops, no check for failure!
printf("Buf is now %s\n", buf); // Undefined Behavior if fgets() returned
NULL
```

Reviewers: svoboda

176. The n parameter is negative or zero for a call to fgets or fgetws. (7.23.7.2, 7.31.3.2).

```
const int buf_size = -1;      // Oops, should be > 0!
char buf[buf_size];
FILE *f = fopen("foo", "r");
if (f == NULL) {
  printf("Can't open foo\n");
}
fgets( buf, buf_size, f);     // Undefined Behavior
```

Reviewers: svoboda

177. The file position indicator for a binary stream is used after a call to the ungetc function where its value was zero before the call (7.23.7.10).

```
FILE *f = fopen("foo", "rb");
if (f == NULL) {
  printf("Can't open foo\n");
}
assert(ftell( f) == 0);
int c = 'A';
c = ungetc( c, f);   // Undefined Behavior
```

Reviewers: svoboda

178. The file position indicator for a stream is used after an error occurred during a call to the fread or fwrite function (7.23.8.1, 7.23.8.2).

```
FILE *f = fopen("foo", "rb");
if (f == NULL) {
  printf("Can't open foo\n");
}
const int buf_size = 125;
```

```c
char buffer[buf_size];
size_t bytes = fread( buffer, 1, buf_size, f);
printf("Read %ld bytes\n", bytes);
if (bytes < buf_size) {   // uh-oh, error
  long pos = ftell( f);   // Undefined Behavior if fread() had error
  printf("File is at %lu position \n", pos);
}
```

Reviewers: svoboda

179. A partial element read by a call to the fread function is used (7.23.8.1).

```c
FILE *f = fopen("foo", "rb");
if (f == NULL) {
  printf("Can't open foo\n");
}
const int buf_size = 250;
wchar_t buffer[buf_size];
for (size_t i = 0; i < buf_size; i++) {
  buffer[i] = 0;
}
size_t bytes = fread( buffer, sizeof(wchar_t), buf_size, f);
printf("%lu bytes read\n", bytes);
if (bytes < buf_size) { // hmmm, less bytes read than expected
  printf("Possible partial byte read is %.02x, %c\n", buffer[bytes],
buffer[bytes]);
  // Undefined Behavior if buffer has partially-read character
}
```

Reviewers: svoboda

180. The fseek function is called for a text stream with a nonzero offset and either the offset was not returned by a previous successful call to the ftell function on a stream associated with the same file or whence is not SEEK_SET (7.23.9.2).

```c
FILE *f = fopen("foo", "r");
if (f == NULL) {
  printf("Can't open foo\n");
}
fseek(f, 5, SEEK_CUR);  // Undefined behavior
```

Reviewers: svoboda

181. The fsetpos function is called to set a position that was not returned by a previous successful call to the fgetpos function on a stream associated with the same file (7.23.9.3).

```c
FILE *opener(const char *filename) {
  fpos_t offset;
```

```
  if (filename == NULL) {
    // ...
  }

  FILE *file = fopen(filename, "r");
  if (file == NULL) {
    // ...
  }

  memset(&offset, 0, sizeof(offset));
  if (fsetpos(file, &offset) != 0) {  // Undefined Behavior
    // ...
  }

  return file;
}
```

Cite: TS17961 5.41 [xfilepos] EXAMPLE

Reviewers: svoboda

182. A non-null pointer returned by a call to the calloc, malloc, realloc, or aligned_alloc function with a zero requested size is used to access an object (7.24.3).

```
#include <stdlib.h>
#include <assert.h>

size_t size = 0;
int *array = malloc(size * sizeof(int));
assert(array);
array[0] = 123;  // Undefined Behavior, out-of-bounds write
```

Reviewers: svoboda, j.myers

183. The value of a pointer that refers to space deallocated by a call to the free or realloc function is used (7.24.3).

```
struct List { struct List *next; // ... };

void free_list(struct List *head) {
  for (; head != NULL; head = head->next) {  // Undefined Behavior
    free(head);
  }
}
```

Cite: TS17961 5.2 [accfree] EXAMPLE 1,2,3

Reviewers: svoboda

184. The pointer argument to the free or realloc function is unequal to a null pointer and 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 (7.24.3.3, 7.24.3.7).

```
void f(size_t num_elem) {
  int error_condition = 0;
  int *x = malloc(num_elem * sizeof(int));
  if (x == NULL) {
    // ...
  }
  // ...
  if (error_condition == 1) {
    // ...
    free(x);
  }
  // ...
  free(x);   // Undefined Behavior
  x = NULL;
}
```

Cite: TS17961 5.23 [dblfree] EXAMPLE 1, 2, Cite: TS17961 5.34 [xfree] EXAMPLE 1, 2

Reviewers: svoboda

185. The value of the object allocated by the malloc function is used (7.24.3.6).

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

unsigned char *p = malloc(10);
if (p != NULL) {
  printf("p[0] is %c\n", p[0]);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

186. The values of any bytes in a new object allocated by the realloc function beyond the size of the old object are used (7.24.3.7).

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

enum { OLD_SIZE = 10, NEW_SIZE = 20 };

int *resize_array(int *array, size_t count) {
  if (0 == count) {
    return 0;
  }
```

```
  int *ret = realloc(array, count * sizeof(int));
  if (!ret) {
    free(array);
    return 0;
  }

  return ret;
}

void func(void) {
  int *array = malloc(OLD_SIZE * sizeof(int));
  if (NULL == array) {
    // Handle Error
  }

  for (size_t i = 0; i < OLD_SIZE; ++i) {
    array[i] = i;
  }

  array = resize_array(array, NEW_SIZE);
  if (NULL == array) {
    // Handle Error
  }

  for (size_t i = 0; i < NEW_SIZE; ++i) {
    printf("%d ", array[i]);  // Undefined Behavior on new array members
  }
}
```

Cite: CERT C Rule EXP33-C 5th NCCE 4.3.11

Reviewers: svoboda

187. The program calls the exit or quick_exit function more than once, or calls both functions (7.24.4.4, 7.24.4.7).

```
#include <stdlib.h>

void exit1(void) {
  // Cleanup code ...
  return;
}

void exit2(void) {
  extern int some_condition;
  if (some_condition) {
    // More cleanup code ...
    exit(0);  // Undefined Behavior when invoked within exit()
```

```
  }
  return;
}

int main(void) {
  if (atexit(exit1) != 0) {
    // Handle Error
  }
  if (atexit(exit2) != 0) {
    // Handle Error
  }
  // Program code ...
  return 0;
}
```

Cite: CERT C Rule ENV32-C 1st NCCE 11.3.1

Reviewers: svoboda

188. During the call to a function registered with the atexit or at_quick_exit function, a call is made to the longjmp function that would terminate the call to the registered function (7.24.4.4, 7.24.4.7).

```
#include <stdlib.h>
#include <setjmp.h>

jmp_buf env;
int val;

void exit1(void) {
  longjmp(env, 1);  // Undefined Behavior
}

int main(void) {
  if (atexit(exit1) != 0) {
    // Handle Error
  }
  if (setjmp(env) == 0) {
    exit(0);
  } else {
    return 0;
  }
}
```

Cite: CERT C Rule ENV32-C 2nd NCCE 11.3.3

Reviewers: svoboda

189. The string set up by the getenv or strerror function is modified by the program (7.24.4.6, 7.26.6.3).

```
void f3(void) {
  char *shell_dir = getenv("SHELL");

  if (shell_dir != NULL) {
    char *slash = strrchr(shell_dir, '/');
    if (slash) {
      *slash = '\0';  // Undefined Behavior
    }
    // Use shell_dir...
  }
}
```

Cite: TS17961 5.29 [libmod] EXAMPLE 3, 4

Reviewers: svoboda

190. A signal is raised while the quick_exit function is executing (7.24.4.7).

```
void exit_handler(void) {
  raise(SIGINT);   // Undefined Behavior
}

int main(void) {
  if (atexit(exit_handler) != 0) {
    // Handle Error
  }
  quick_exit(0);
  return 0;
}
```

Reviewers: svoboda

191. A command is executed through the system function in a way that is documented as causing termination or some other form of undefined behavior (7.24.4.8).

HYPOTHETICAL COMPILABLE EXAMPLE?

```
int retval = system("ls /missing");   // Undefined Behavior
```

Reviewers: svoboda

192. A searching or sorting utility function is called with an invalid pointer argument, even if the number of elements is zero (7.24.5).

```
#include <stdlib.h>

int compare(const void *a, const void *b) {
```

```
    return (*(int *)a - *(int *)b);
}

int *arr = NULL;
qsort(arr, 0, sizeof(int), compare);  // Undefined Behavior
```

Reviewers: coates, svoboda

193. The comparison function called by a searching or sorting utility function alters the contents of the array being searched or sorted, or returns ordering values inconsistently (7.24.5).

```
#include <stdlib.h>

int compare(const void *a, const void *b) {
    *(int *)a = *(int *)a + 1;        // Modify array contents!
    return (*(int *)a - *(int *)b);
}

int arr[] = {2, 1, 4, 1, 5, 9, 2, 6};
size_t n = sizeof(arr) / sizeof(arr[0]);
qsort(arr, n, sizeof(int), compare);  // Undefined Behavior
```

Reviewers: coates, svoboda

194. The array being searched by the bsearch function does not have its elements in proper order (7.24.5.1).

```
int compare(const void *a, const void *b) {
  return *(int *)a - *(int *)b;
}

int arr[] = {2, 3, 5, 11, 7}; // Oops, mis-ordered array!
int n = sizeof(arr)/sizeof(arr[0]);
int key = 7;
int *result = (int *)bsearch(&key, arr, n, sizeof(int), compare);
// Undefined Behavior
```

Reviewers: svoboda

195. The current conversion state is used by a multibyte/wide character conversion function after changing the LC_CTYPE category (7.24.7).

EXAMPLE: fscanf() is defined in C23 s7.23.6.2. Example 6, (paragraph 23) in this section describes a hypothetical encoding with shift states, which we will use for this example:

```
#include <stdio.h>
#include <stdlib.h>
#include <wchar.h>
#include <memory.h>
```

```c
#include <locale.h>

char str[50];
wchar_t wstr[50];
memset(wstr, 0, sizeof(wstr));
int counter = 0;
setlocale(LC_CTYPE, "HYPO");                // Use hypothetical encoding
// Suppose standard input contains the single line: ↑□X□Y↓
fgets(str, sizeof(str), stdin);             // str == "↑□X□Y↓"
counter += mbtowc(wstr, str, 4);            // wstr == "□X", upper state
setlocale(LC_CTYPE, "C");                   // state changed
counter += mbtowc(wstr, &str[counter], 4);  // Undefined Behavior
```

Reviewers: svoboda, j.myers

196. A string or wide string utility function is instructed to access an array beyond the end of an object (7.26.1, 7.31.4).

```c
#include <stddef.h>
#include <string.h>

void func(void) {
  wchar_t wide_str1[] = L"0123456789";
  wchar_t wide_str2[] = L"0000000000";

  strncpy(wide_str2, wide_str1, 10);  // Undefined Behavior
}
```

Cite: CERT C Rule STR38-C 1st NCCE 8.6.1, 2nd NCCE 8.6.2, 3rd NCCE 8.6.4

Reviewers: svoboda

197. A string or wide string utility function is called with an invalid pointer argument, even if the length is zero (7.26.1, 7.31.4).

```c
#include <string.h>

char *c = 0;
int length = strlen(c); // Undefined Behavior
```

Reviewers: svoboda, j.myers

198. The contents of the destination array are used after a call to the strxfrm, strftime, wcsxfrm, or wcsftime function in which the specified length was too small to hold the entire null-terminated result (7.26.4.5, 7.29.3.5, 7.31.4.4.4, 7.31.5.1).

```c
char src[] = "This is a test";
const int string_size = strlen(src) + 1;
```

```
char dest[string_size - 1];                    // Oops, too small!
int length = strxfrm(dest, src, string_size);  // Undefined Behavior
```

Reviewers: svoboda

199. A sequence of calls of the strtok function is made from different threads (7.26.5.9).

```
#include <string.h>
#include <threads.h>

int bar(void *) {
  char *t = strtok(NULL, "#,"); // Undefined Behavior
  return t[0];
}


char str[] = "?a???b,,,#c";
char *t = strtok(str, "?");
thrd_t thr;
if (thrd_success != thrd_create(&thr, bar, 0)) {
  // Handle Error
}

t = strtok(NULL, ","); // Undefined Behavior

int retval;
if (thrd_success != thrd_join(thr, &retval)) {
  // Handle Error
}
```

Reviewers: svoboda, CliveP, robin-rowe, j.myers

200. The first argument in the very first call to the strtok or wcstok is a null pointer (7.26.5.9, 7.31.4.5.8).

```
#include <string.h>

static char str[] = "?a???b,,,#c";
char *t;
t = strtok(NULL, "?");  // Undefined Behavior
```

Reviewers: svoboda, j.myers

201. A pointer returned by the strerror function is used after a subsequent call to the function, or after the calling thread has exited (7.26.6.3).

```
#include <stdio.h>
#include <errno.h>
```

```c
char *inval = strerror(EINVAL);
char *perm = strerror(EPERM);
printf("Invalid: %s\n", inval); // Undefined Behavior, invalidated by perm
printf("Permission: %s\n", perm);
```

Reviewers: svoboda, j.myers

202. The type of an argument to a type-generic macro is not compatible with the type of the corresponding parameter of the selected function (7.27).

```c
double complex dc = CMPLX( 2, 3); // 2 + 3i
double result = dsqrt(dc);        // Undefined Behavior
```

Reviewers: svoboda

203. Arguments for generic parameters of a type-generic macro are such that some argument has a corresponding real type that is of standard floating type and another argument is of decimal floating type (7.27).

```c
double d = 3.0;
_Decimal64 d64 = 2.0;
double result = remainder(d64, d);   // Undefined Behavior
```

Reviewers: svoboda

204. Arguments for generic parameters of a type-generic macro are such that neither <math.h> and <complex.h> define a function whose generic parameters have the determined corresponding real type (7.27).

```c
_Float32 f = 2.0;
long double d = 3.0;
double result = nexttoward(f, d);   // Undefined Behavior
```

Reviewers: svoboda

205. A complex argument is supplied for a generic parameter of a type-generic macro that has no corresponding complex function (7.27).

```c
float complex fc = CMPLX(2, 3); // 2 + 3i
double result = ceil(fc);       // Undefined Behavior
```

Reviewers: svoboda

206. A decimal floating argument is supplied for a generic parameter of a type-generic macro that expects a complex argument (7.27).

```c
_Decimal64 d64 = 2.0;
double result = carg(d64);   // Undefined Behavior
```

Reviewers: svoboda

207. A standard floating or complex argument is supplied for a generic parameter of a type-generic macro that expects a decimal floating type argument (7.27).

```
double d = 3.0;
_Decimal64 d64 = 2.0;
double result = ddiv(d64, d);   // Undefined Behavior
```

Reviewers: svoboda

208. A non-recursive mutex passed to mtx_lock is locked by the calling thread (7.28.4.3).

```
#include <threads.h>

mtx_t m;
if (thrd_success != mtx_init(&m, mtx_plain)) {
  // Handle Error
}

if (thrd_success != mtx_lock(&m)) {
  // Handle Error
}

if (thrd_success != mtx_lock(&m)) {  // Undefined Behavior
  // Handle Error
}

mtx_destroy(&m);
```

Reviewers: svoboda, j.myers

209. The mutex passed to mtx_timedlock does not support timeout (7.28.4.4).

```
#include <threads.h>
#include <time.h>

mtx_t m;
if (thrd_success != mtx_init(&m, mtx_plain)) { // Oops, should be mtx_timed!
  // Handle Error
}

struct timespec ts;
if (0 == timespec_get(&ts, TIME_UTC)) {
  // Handle Error
}
ts.tv_sec += 1; // 1 second from now

if (thrd_success != mtx_timedlock(&m, &ts)) { // Undefined Behavior
  // Handle Error
```

```
}

mtx_destroy(&m);
```

Reviewers: svoboda, j.myers

210. The mutex passed to mtx_unlock is not locked by the calling thread (7.28.4.6).

```
#include <threads.h>

mtx_t m;
if (thrd_success != mtx_init(&m, mtx_plain)) {
  // Handle Error
}

if (thrd_success != mtx_unlock(&m)) {   // Undefined Behavior
  // Handle Error
}

mtx_destroy(&m);
```

Reviewers: svoboda, j.myers

211. The thread passed to thrd_detach or thrd_join was previously detached or joined with another thread (7.28.5.3, 7.28.5.6).

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

int thread_func(void *arg) {
  // Do work ...
  thrd_detach(thrd_current());
  return 0;
}

int main(void) {
  thrd_t t;

  if (thrd_success != thrd_create(&t, thread_func, NULL)) {
    // Handle Error
    return 0;
  }

  if (thrd_success != thrd_join(t, 0)) {  // Undefined Behavior
    // Handle Error
    return 0;
  }
  return 0;
}
```

Cite: CERT C Rule CON39-C 1st NCCE 14.10.1

Reviewers: svoboda

212. The tss_create function is called from within a destructor (7.28.6.1).

```c
#include <threads.h>

void destructor(void *arg) {
  tss_t key;
  if (thrd_success != tss_create(&key, 0)) { // Undefined Behavior
    // Handle Error
  }
}

int func(void *) {
  tss_t key;
  if (thrd_success != tss_create(&key, destructor)) {
    // Handle Error
  }
  static char str[] = "Hello";
  tss_set(key, str);
  return 0;
}

int foo(void *) {
  thrd_t thr;
  if (thrd_success != thrd_create(&thr, func, 0)) {
    // Handle Error
  }

  int retval;
  if (thrd_success != thrd_join(thr, &retval)) {
    // Handle Error
  }
  return 0;
}
```

Reviewers: svoboda, j.myers

213. The key passed to tss_delete, tss_get, or tss_set was not returned by a call to tss_create before the thread commenced executing destructors (7.28.6.2, 7.28.6.3, 7.28.6.4).

```c
#include <threads.h>

void destructor(void *arg) {
  tss_t key;
  if (thrd_success != tss_create(&key, 0)) { // Undefined Behavior
    // Handle Error
```

```
  }
}

int func(void *) {
  tss_t key;
  static char str[] = "Hello";
  tss_set(key, str); // Undefined Behavior, key not initialized by
tss_create()
  return 0;
}

int foo(void *) {
  thrd_t thr;
  if (thrd_success != thrd_create(&thr, func, 0)) {
    // Handle Error
  }

  int retval;
  if (thrd_success != thrd_join(thr, &retval)) {
    // Handle Error
  }
  return 0;
}
```

Reviewers: svoboda, j.myers

214. An attempt is made to access the pointer returned by the time conversion functions after the thread that originally called the function to obtain it has exited (7.29.3).

```
#include <stdio.h>
#include <threads.h>
#include <time.h>

char *now = 0;

int bar(void *) {
  time_t n1;
  if ((time_t) -1 == time(&n1)) {
    // Handle Error
  }
  struct tm *n2 = localtime(&n1);
  now = asctime(n2);
  return 0;
}

void foo(void) {
  thrd_t thr;
  if (thrd_success != thrd_create(&thr, bar, 0)) {
```

```
    // Handle Error
  }

  int retval;
  if (thrd_success != thrd_join(thr, &retval)) {
    // Handle Error
  }

  printf("The time is %s\n", now); // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

215. At least one member of the broken-down time passed to asctime contains a value outside its normal range, or the calculated year exceeds four digits or is less than the year 1000 (7.29.3.1).

```
#include <time.h>

void func(struct tm *time_tm) {
  char *time = asctime(time_tm);  // Undefined Behavior
  // ...
}
```

Cite: CERT C Rule MSC33-C 1st NCCE 15.3.1

Reviewers: svoboda

216. The argument corresponding to an s specifier without an l qualifier in a call to the fwprintf function does not point to a valid multibyte character sequence that begins in the initial shift state (7.31.2.11).

```
#include <wchar.h>
#include <locale.h>

setlocale(LC_ALL, "UTF-8");
// In UTF-8: the Euro symbol == '€' == U+20AC == \xE2 \x82 \xAC == \342 \202
\254
const char invalid[] = {'\xE2', '\0'};   // invalid UTF-8
fwprintf(stdout, L"The string is %ls\n", invalid);
// Undefined Behavior in UTF-8 locale
```

Reviewers: svoboda, j.myers

217. In a call to the wcstok function, the object pointed to by ptr does not have the value stored by the previous call for the same wide string (7.31.4.5.8).

```
#include <wchar.h>
#include <stdio.h>
```

```c
wchar_t input[20] = L"foo bar baz";
wchar_t *buffer;
wchar_t *token = wcstok(input, L" ", &buffer);
while (token) {
  wprintf(L"%ls\n", token);
  wcscpy(buffer, input);              // buffer changed
  token = wcstok(0, L" ", &buffer);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

218. An mbstate_t object is used inappropriately (7.31.6).

```c
#include <string.h>
#include <wchar.h>

void func(const char *mbs) {
  size_t len;
  mbstate_t state;
  len = mbrlen(mbs, strlen(mbs), &state);  // Undefined Behavior
}
```

Cite: CERT C Rule EXP33-C 3rd NCCE 4.3.7

Reviewers: svoboda

219. The value of an argument of type wint_t to a wide character classification or case mapping function is neither equal to the value of WEOF nor representable as a wchar_t (7.32.1).

```c
#include <wctype.h>

int main(void) {
  int flag = iswalpha(WINT_MIN);
  // Undefined Behavior if sizeof(wchar_t) < sizeof(wint_t)
```

Reviewers: svoboda, j.myers

220. The iswctype function is called using a different LC_CTYPE category from the one in effect for the call to the wctype function that returned the description (7.32.2.2.1).

See UB 191 for background on LC_CTYPE categories. (editor; UB 191 of which version?)

```c
#include <wchar.h>
#include <locale.h>

int f(wint_t wc) {
  wctype_t alpha = wctype("alpha");
```

```
  setlocale(LC_CTYPE, "C");     // state changed
  return iswctype(wc, alpha);   // Undefined Behavior
}
```

Reviewers: svoboda, j.myers

221. The towctrans function is called using a different LC_CTYPE category from the one in effect for the call to the wctrans function that returned the description (7.32.3.2.1).

```
#include <wchar.h>
#include <wctype.h>
#include <locale.h>

wint_t f(wint_t wc) {
  wctype_t lower = wctrans("tolower");
  setlocale(LC_CTYPE, "C");     // state changed
  return towctrans(wc, lower);  // Undefined Behavior
}
```

Reviewers: svoboda, j.myers