EXP44-C. Do not rely on side effects in operands to sizeof, _Alignof, or _Generic

Some operators do not evaluate their operands beyond the type information the operands provide. When using one of these operators, do not pass an operand that would otherwise yield a side effect since the side effect will not be generated.

The sizeof operator yields the size (in bytes) of its operand, which may be an expression or the parenthesized name of a type. In most cases, the operand is not evaluated. A possible exception is when the type of the operand is a variable length array type (VLA); then the expression is evaluated. When part of the operand of the sizeof operator is a VLA type and when changing the value of the VLA’s size expression would not affect the result of the operator, it is unspecified whether or not the size expression is evaluated. (See unspecified behavior 22.)

The operand passed to _Alignof is never evaluated, despite not being an expression. For instance, if the operand is a VLA type and the VLA’s size expression contains a side effect, that side effect is never evaluated.

The operand used in the controlling expression of a _Generic selection expression is never evaluated.

Providing an expression that appears to produce side effects may be misleading to programmers who are not aware that these expressions are not evaluated, and in the case of a VLA used in sizeof, have unspecified results. As a result, programmers may make invalid assumptions about program state, leading to errors and possible software vulnerabilities.

This rule is similar to PRE31-C. Avoid side effects in arguments to unsafe macros.

Noncompliant Code Example (sizeof)

In this noncompliant code example, the expression a++ is not evaluated:

```c
#include <stdio.h>
void func(void) {
  int a = 14;
  int b = sizeof(a++);
  printf("%d, %d\n", a, b);
}
```

Consequently, the value of a after b has been initialized is 14.

Compliant Solution (sizeof)

In this compliant solution, the variable a is incremented outside of the sizeof operation:

```c
#include <stdio.h>
void func(void) {
  int a = 14;
  int b = sizeof(a);
  ++a;
  printf("%d, %d\n", a, b);
}
```

Noncompliant Code Example (sizeof, VLA)

In this noncompliant code example, the expression ++n in the initialization expression of a must be evaluated because its value affects the size of the VLA operand of the sizeof operator. However, in the initialization expression of b, the expression ++n % 1 evaluates to 0. This means that the value of n does not affect the result of the sizeof operator. Consequently, it is unspecified whether or not n will be incremented when initializing b.
#include <stddef.h>
#include <stdio.h>

void f(size_t n) {
    /* n must be incremented */
    size_t a = sizeof(int[++n]);

    /* n need not be incremented */
    size_t b = sizeof(int[++n % 1 + 1]);

    printf("%zu, %zu, %zu\n", a, b, n);
    /* ... */
}

Compliant Solution (sizeof, VLA)

This compliant solution avoids changing the value of the variable `n` used in each `sizeof` expression and instead increments `n` safely afterwards:

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

void f(size_t n) {
    size_t a = sizeof(int[n + 1]);
    ++n;

    size_t b = sizeof(int[n % 1 + 1]);
    ++n;
    printf("%zu, %zu, %zu\n", a, b, n);
    /* ... */
}
```

Noncompliant Code Example (_Generic)

This noncompliant code example attempts to modify a variable’s value as part of the `_Generic` selection control expression. The programmer may expect `a` is incremented, but because `_Generic` does not evaluate its control expression, the value of `a` is not modified.

```c
#include <stdio.h>
#define S(val) _Generic(val, int : 2, 
    short : 3, 
    default : 1)

void func(void) {
    int a = 0;
    int b = S(a++);
    printf("%d, %d\n", a, b);
}
```

Compliant Solution (_Generic)

In this compliant solution, `a` is incremented outside of the `_Generic` selection expression:
```c
#include <stdio.h>

#define S(val) _Generic(val, int : 2, 
    short : 3, 
    default : 1)

void func(void) {
    int a = 0;
    int b = S(a);
    ++a;
    printf("%d, %d\n", a, b);
}
```

**Noncompliant Code Example (\_Alignof)**

This noncompliant code example attempts to modify a variable while getting its default alignment value. The user may have expected \_Aligned to be incremented as part of the \_Alignof expression, but because \_Alignof does not evaluate its operand, \_Aligned is unchanged.

```c
#include <stdio.h>

void func(void) {
    int _Aligned = 0;
    /* ... */
    int _Aligned = _Alignof(int[++val]);
    printf("%zu, %d\n", align, val);
    /* ... */
}
```

**Compliant Solution (\_Alignof)**

This compliant solution moves the expression out of the \_Alignof operator:

```c
#include <stdio.h>

void func(void) {
    int val = 0;
    /* ... */
    ++val;
    size_t align = _Alignof(int[val]);
    printf("%zu, %d\n", align, val);
    /* ... */
}
```

**Exceptions**

EXP44-C-EX1: Reading a volatile-qualified value is a side-effecting operation. However, accessing a value through a volatile-qualified type does not guarantee side effects will happen on the read of the value unless the underlying object is also volatile-qualified. Idiomatic reads of a volatile-qualified object are permissible as an operand to a sizeof(), _Alignof(), or _Generic expression, as in the following example:

```c
void f(void) {
    int volatile v;
    (void)sizeof(*v);
}
```

**Risk Assessment**

If expressions that appear to produce side effects are supplied to an operator that does not evaluate its operands, the results may be different than expected. Depending on how this result is used, it can lead to unintended program behavior.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP44-C</td>
<td>Low</td>
<td>Unlikely</td>
<td>Low</td>
<td>P3</td>
<td>L3</td>
</tr>
</tbody>
</table>
Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrée</td>
<td>19.04</td>
<td>alignof-side-effect generic-selection-side-effect sizeof</td>
<td>Fully checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-EXP44</td>
<td></td>
</tr>
<tr>
<td>Clang</td>
<td>3.9</td>
<td>-Wunevaluated-expression</td>
<td>Can diagnose some instance of this rule, but not all (such as the _Alignof NCCE).</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>LANG_STRUCT.SE.SIZEOF</td>
<td>Side effects in sizeof</td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coverity</td>
<td>2017.07</td>
<td>MISRA C 2004 Rule 12.3</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.EXP06</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>MISRA.SIZEOF.SIDE_EFFECT</td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>54 S, 653 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-EXP44-a CERT_C-EXP44-b</td>
<td>Object designated by a volatile lvalue should not be accessed in the operand of the sizeof operator. The function call that causes the side effect shall not be the operand of the sizeof operator</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C: Rule EXP44-C</td>
<td>Checks for situations when side effects of specified expressions are ignored (rule fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>3307</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>7.07</td>
<td>V568</td>
<td></td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>alignof-side-effect generic-selection-side-effect sizeof</td>
<td>Fully checked</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

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

Related Guidelines

Key here (explains table format and definitions)