FLP36-C. Preserve precision when converting integral values to floating-point type

Narrower arithmetic types can be cast to wider types without any effect on the magnitude of numeric values. However, whereas integer types represent exact values, floating-point types have limited precision. The C Standard, 6.3.1.4 paragraph 2 [ISO/IEC 9899:2011], states

> When a value of integer type is converted to a real floating type, if the value being converted can be represented exactly in the new type, it is unchanged. If the value being converted is in the range of values that can be represented but cannot be represented exactly, the result is either the nearest higher or nearest lower representable value, chosen in an implementation-defined manner. If the value being converted is outside the range of values that can be represented, the behavior is undefined. Results of some implicit conversions may be represented in greater range and precision than that required by the new type (see 6.3.1.8 and 6.8.6.4).

Conversion from integral types to floating-point types without sufficient precision can lead to loss of precision (loss of least significant bits). No runtime exception occurs despite the loss.

Noncompliant Code Example

In this noncompliant example, a large value of type `long int` is converted to a value of type `float` without ensuring it is representable in the type:

```c
#include <stdio.h>

int main(void) {
    long int big = 1234567890L;
    float approx = big;
    printf("%ld\n", (big - (long int)approx));
    return 0;
}
```

For most floating-point hardware, the value closest to 1234567890 that is representable in type `float` is 1234567844; consequently, this program prints the value -46.

Compliant Solution

This compliant solution replaces the type `float` with a `double`. Furthermore, it uses an assertion to guarantee that the `double` type can represent any `long int` without loss of precision. (See INT35-C. Use correct integer precisions and MSC11-C. Incorporate diagnostic tests using assertions.)

```c
#include <assert.h>
#include <float.h>
#include <limits.h>
#include <math.h>
#include <stdint.h>
#include <stdio.h>

extern size_t popcount(uintmax_t); /* See INT35-C */
#define PRECISION(umax_value) popcount(umax_value)

int main(void) {
    assert(PRECISION(LONG_MAX) <= DBL_MANT_DIG * log2(FLT_RADIX));
    long int big = 1234567890L;
    double approx = big;
    printf("%ld\n", (big - (long int)approx));
    return 0;
}
```

On the same implementation, this program prints 0, implying that the integer value 1234567890 is representable in type `double` without change.

Risk Assessment

Conversion from integral types to floating-point types without sufficient precision can lead to loss of precision (loss of least significant bits).
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></td>
<td>Supported: Astrée keeps track of all floating point rounding errors and loss of precision and reports code defects resulting from those.</td>
</tr>
<tr>
<td>Coverity</td>
<td>2017.07</td>
<td>MISRA C 2004 Rule 10.x (needs investigation)</td>
<td>Needs investigation</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>435 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++ test</td>
<td>10.4.2</td>
<td>CERT_C-FLP36-a, CERT_C-FLP36-b</td>
<td>Implicit conversions from integral to floating type which may result in a loss of information shall not be used. Implicit conversions from integral constant to floating type which may result in a loss of information shall not be used.</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT-C: Rule FLP36-C</td>
<td>Checks for precision loss in integer to float conversion (rule fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>1260, 1263, 1298, 1299, 1800, 1802, 1803, 1804, 4117, 4435, 4437, 4445</td>
<td></td>
</tr>
<tr>
<td>PRQA QA-C++</td>
<td>4.4</td>
<td>3011</td>
<td></td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>7.07</td>
<td>V674</td>
<td></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)

<table>
<thead>
<tr>
<th>Taxonomy</th>
<th>Taxonomy item</th>
<th>Relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>CERT C Secure Coding Standard</td>
<td>DCL03-C. Use a static assertion to test the value of a constant expression</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
<tr>
<td>CERT Oracle Secure Coding Standard for Java</td>
<td>NUM13-J. Avoid loss of precision when converting primitive integers to floating-point</td>
<td>Prior to 2018-01-12: CERT: Unspecified Relationship</td>
</tr>
</tbody>
</table>

Bibliography