FLP06-C. Convert integers to floating point for floating-point operations

Using integer arithmetic to calculate a value for assignment to a floating-point variable may lead to loss of information. This problem can be avoided by converting one of the integers in the expression to a floating type.

When converting integers to floating-point values, and vice versa, it is important to carry out proper range checks to avoid undefined behavior (see FLP34-C. Ensure that floating-point conversions are within range of the new type).

Noncompliant Code Example

In this noncompliant code example, the division and multiplication operations take place on integers and are then converted to floating point. Consequently, floating-point variables `d`, `e`, and `f` are not initialized correctly because the operations take place before the values are converted to floating-point values. The results are truncated to the nearest integer or may overflow.

```
void func(void) {
  short a = 533;
  int b = 6789;
  long c = 466438237;

  float d = a / 7; /* d is 76.0 */
  double e = b / 30; /* e is 226.0 */
  double f = c * 789; /* f may be negative due to overflow */
}
```

Compliant Solution (Floating-Point Literal)

In this compliant solution, the decimal error in initialization is eliminated by ensuring that at least one of the operands to the division operation is floating point:

```
void func(void) {
  short a = 533;
  int b = 6789;
  long c = 466438237;

  float d = a / 7.0f; /* d is 76.14286 */
  double e = b / 30.; /* e is 226.3 */
  double f = (double)c * 789; /* f is 368019768993.0 */
}
```

Compliant Solution (Conversion)

In this compliant solution, the decimal error in initialization is eliminated by first storing the integer in the floating-point variable and then performing the arithmetic operation. This practice ensures that at least one of the operands is a floating-point number and that the subsequent arithmetic operation is performed on floating-point operands.

```
void func(void) {
  short a = 533;
  int b = 6789;
  long c = 466438237;

  float d = a;
  double e = b;
  double f = c;

  d /= 7; /* d is 76.14286 */
  e /= 30; /* e is 226.3 */
  f *= 789; /* f is 368019768993.0 */
}
```

Exceptions

FLP06-C-EX0: It may be desirable to have the operation take place as integers before the conversion (obviating the need for a call to `trunc()`, for example). If this is the programmer's intention, it should be clearly documented to help future maintainers understand that this behavior is intentional.

Risk Assessment

Improper conversions between integers and floating-point values may yield unexpected results, especially loss of precision. Additionally, these unexpected results may actually involve overflow, or undefined behavior.

Recommendation	Severity	Likelihood	Remediation Cost	Priority	Level
FLP06-C	Low	Probable	Low	P6	L2

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04		Supported: This rule aims to prevent truncations and overflows. All possible overflows are reported by Astrée.
Axivion Bauhaus Suite	6.9.0	**CertC-FLP06**	
CodeSonar	5.1p0	**LANG.TYPE.MOT**	Mismatched operand types
Compass/ROSE			Can detect violations of this rule. Any assignment operation where the type of the assigned-to value is `float` or `double`, but all the expressions to the right of the assignment are integral, is a violation of this rule
LDRA tool suite	9.7.1	**435 S**	Enhanced enforcement
Parasoft C/C++test	10.4.2	**CERT_C-FLP06-a** **CERT_C-FLP06-b**	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
Polyspace Bug Finder	R2019a	CERT C: Rec. FLP06-C	Checks for float overflow (rec. partially covered)
PRQA QA-C	9.5	**4117** **4118**	Partially implemented
PVS-Studio	6.23	**V636**	
Splint	3.1.1		

Related Vulnerabilities

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

Related Guidelines

CERT C Secure Coding Standard	FLP34-C. Ensure that floating-point conversions are within range of the new type
SEI CERT C++ Coding Standard	VOID FLP05-CPP. Convert integers to floating point for floating point operations
CERT Oracle Secure Coding Standard for Java	NUM50-J. Convert integers to floating point for floating-point operations
MITRE CWE	CWE-681, Incorrect conversion between numeric types CWE-682, Incorrect calculation

Bibliography

[Hatton 1995]	Section 2.7.3, "Floating-Point Misbehavior"