ISO/IEC JTC 1/SC 22/WG 23 N 0321

Meeting #17 markup of Proposed separation of XYY into two descriptions

Date	23 March 2011
Contributed by	Secretary
Original file name	
Notes	Replaces N0305

The text of Action Item #16-12 reads as follows:

> Look at XYY in the main document and both annexes to try to tease apart two vulnerabilities: one concerning arithmetic over/underflow and one concerning performing bit/shift operations on numeric values. In both, note that unsigned and signed arithmetic present two different challenges.

After reading the annexes, I realized that redrafting them will be easy once we settle on the text for the body of the report. So, I'm not including text for annexes at this time.

The proposed text for the body of the report follows:

6.x Arithmetic Wrap-around Error [FIF]

6.x.1 Description of application vulnerability

Wrap-around errors can occur whenever a value is incremented past the maximum or decremented past the minimum value representable in its type and, depending upon

- whether the type is signed or unsigned,
- the specification of the language semantics and/or
- implementation choices,

"wraps around" to an unexpected value. This vulnerability is related to Logical Wrap-around Error [PIK]. This description is derived from Wrap-Around Error [XYY], which appeared in Edition 1 of this international technical report.

Comment [JWM1]: Convert this to a footnote.

6.x.2 Cross reference [Note to editor: Please verify the applicability of these cross-references.]

CWE:

128. Wrap-around Error
190: Integer Overflow or Wraparound

JSF AV Rules: 164 and 15
MISRA C 2004: 10.1 to 10.6, 12.8 and 12.11
MISRA C++ 2008: 2-13-3, 5-0-3 to 5-0-10, and 5-19-1
CERT C guidelines: INT30-C, INT32-C, and INT34-C

6.x.3 Mechanism of failure

Due to how arithmetic is performed by computers, if a variable's value is increased past the maximum value representable in its type, the system may fail to provide an overflow indication to the program. One of the most common processor behaviour is to "wrap" to a very large negative value, or set a condition flag for overflow or underflow, or saturate at the largest representable value.

Wrap-around often generates an unexpected negative value; this unexpected value may cause a loop to continue for a long time (because the termination condition requires a value greater than some positive value) or an array bounds violation. A wrap-around can sometimes trigger buffer overflows that can be used to execute arbitrary code.

It should be noted that the precise consequences of wrap-around differ depending on:

- Whether the type is signed or unsigned
- Whether the type is a modulus type
- Whether the type's range is violated by exceeding the maximum representable value or falling short of the minimum representable value
- The semantics of the language specification
- Implementation decisions

However, in all cases, the resulting problem is that the value yielded by the computation may be unexpected.

6.x.4 Applicable language characteristics

This vulnerability description is intended to be applicable to languages with the following characteristics:

- Languages that do not trigger an exception condition when a wrap-around error occurs.

6.x.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Determine applicable upper and lower bounds for the range of all variables and use language mechanisms or static analysis to determine that values are confined to the proper range.
- Analyze the software using static analysis looking for unexpected consequences of arithmetic operations.

6.x.6 Implications for standardization

In future standardization activities, the following items should be considered:

- Language standards developers should consider providing facilities to specify either an error, a saturated value, or a modulo result when numeric overflow occurs. Ideally, the selection among these alternatives could be made by the programmer.

6.y ~~Logical Wrap-around Error~~Using Shift Operations for Multiplication and Division [PIK]

6.y.1 Description of application vulnerability

Using shift operations as a surrogate for multiply or divide may produce an unexpected value when significant bits are lost. This vulnerability is related to Arithmetic Wrap-around Error [FIF]. This description is derived from Wrap-Around Error [XYY], which appeared in Edition 1 of this international technical report.

Comment [JWM2]: Rephrase as shiftin into sign bit or losing value bits. You din;t want to lose value bits or change the sign bit.

Comment [JWM3]: Convert to footnote.

6.x.2 Cross reference [Note to editor: Please verify the applicability of these items.]

CWE:

128. Wrap-around Error
190: Integer Overflow or Wraparound

JSF AV Rules: 164 and 15
MISRA C 2004: 10.1 to 10.6, 12.8 and 12.11
MISRA C++ 2008: 2-13-3, 5-0-3 to 5-0-10, and 5-19-1
CERT C guidelines: INT30-C, INT32-C, and INT34-C

6.y.3 Mechanism of failure

~~Coders sometimes use shift operations with the intention of producing results equivalent to multiplying by a power of two or dividing by a power of two. However, errors can result from this practice~~Shift operations intended to produce results equivalent to multiplication or division fail to produce correct results if the shift operation affects the sign bit or shifts significant bits from the value. ~~For example, if the programmer mistakenly uses logical shifts on signed arithmetic values, the results may test correctly for small values but produce unexpected results when used with large values. The problem, of course, is that the sign bit can be shifted out of the value converting a negative value into a positive one or vice versa.~~

~~Even when the correct type of shift is coded, there can still be problems with unexpected and undetected numerical underflow or overflow if significant bits are shifted out of the value.~~

~~Stated most generally, replacing multiply and divide operations with shifting operations requires detailed knowledge of the representation of the values across the varieties of processors on which the code may be used. In addition, it requires detailed analysis of the range of values for which the shift operations will produce valid results and checking (or static analysis) to ensure that the values never go outside of the range.~~

~~Wrap-around~~Such errors often generate~~s~~ an unexpected negative value; this unexpected value may cause a loop to continue for a long time (because the termination condition requires a value greater than some positive value) or an array bounds violation. ~~A wrap-around~~The error can sometimes trigger buffer overflows that can be used to execute arbitrary code.

6.y.4 Applicable language characteristics

This vulnerability description is intended to be applicable to languages with the following characteristics:

- ~~Languages that do not trigger an exception condition when a wrap-around error occurs.~~

- ~~Languages that do not fully specify the distinction between arithmetic and logical shifts.~~ Languages that permit logical shift operations on variables of arithmetic type.

6.y.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Determine applicable upper and lower bounds for the range of all variables and use language mechanisms or static analysis to determine that values are confined to the proper range.
- Analyze the software using static analysis looking for unexpected consequences of shift operations.
- Avoid using shift operations as a surrogate for multiplication and division. Most compilers will use the correct operation in the appropriate fashion when it is applicable.

6.y.6 Implications for standardization

In future standardization activities, the following items should be considered:

- ~~Language standards developers should consider providing facilities to specify either an error, a saturated value, or a modulo result when logical overflow occurs. Ideally, the selection among these alternatives could be made by the programmer.~~Not providing logical shifting on arithmetic values or flagging it for reviewers.