INT13-C. Use bitwise operators only on unsigned operands

Bitwise operators include the complement operator ~, bitwise shift operators >> and <<, bitwise AND operator &, bitwise exclusive OR operator ^, bitwise inclusive OR operator | and compound assignment operators >>=, <<=, &=, ^= and |=. Bitwise operators should be used only with unsigned integer operands, as the results of bitwise operations on signed integers are implementation-defined.

The C11 standard, section 6.5, paragraph 4 [ISO/IEC 9899:2011], states:

Some operators (the unary operator ~, and the binary operators <<, >>, &, ^, and |), collectively described as bitwise operators) shall have operands that have integral type. These operators return values that depend on the internal representations of integers, and thus have implementation-defined and undefined aspects for signed types.

Furthermore, the bitwise shift operators << and >> are undefined under many circumstances, and are implementation-defined for signed integers for more circumstances; see rule INT34-C. Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand for more information.

Implementation details

The Microsoft C compiler documentation says that:

Bitwise operations on signed integers work the same as bitwise operations on unsigned integers.

On-line GCC documentation about the implementation of bitwise operations on signed integers says:

Bitwise operators act on the representation of the value including both the sign and value bits, where the sign bit is considered immediately above the highest-value bit.

Noncompliant Code Example (Right Shift)

The right-shift operation may be implemented as either an arithmetic (signed) shift or a logical (unsigned) shift. If $E_1$ in the expression $E_1 >> E_2$ has a signed type and a negative value, the resulting value is implementation-defined. Also, a bitwise shift can result in undefined behavior. (See INT34-C. Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand.)

This noncompliant code example can result in an error condition on implementations in which an arithmetic shift is performed, and the sign bit is propagated as the number is shifted [Dowd 2006]:

```c
int rc = 0;
int stringify = 0x80000000;
char buf[sizeof("256")];
rc = snprintf(buf, sizeof(buf), "%u", stringify >> 24);
if (rc == -1 || rc >= sizeof(buf)) {
    /* Handle error */
}
```

In this example, stringify >> 24 evaluates to 0xFFFFFFFF80, or 4,294,967,168. When converted to a string, the resulting value "4294967168" is too large to store in buf and is truncated by snprintf().

If this code had been implemented using sprintf() instead of snprintf(), this noncompliant code example would have resulted in a buffer overflow.

Compliant Solution (Right Shift)

In this compliant solution, stringify is declared as an unsigned integer. The value of the result of the right-shift operation is the integral part of the quotient of stringify / 2 ^ 24:

```c
int rc = 0;
unsigned int stringify = 0x80000000;
char buf[sizeof("256")];
rc = snprintf(buf, sizeof(buf), "%u", stringify >> 24);
if (rc == -1 || rc >= sizeof(buf)) {
    /* Handle error */
}
```

Also, consider using the sprintf_s() function, defined in ISO/IEC TR 24731-1, instead of snprintf() to provide some additional checks. (See STR07-C. Use the bounds-checking interfaces for string manipulation.)
Exceptions

**INT13-C-EX1:** When used as bit flags, it is acceptable to use preprocessor macros or enumeration constants as arguments to the `&` and `|` operators even if the value is not explicitly declared as unsigned.

```c
fd = open(file_name, UO_WRONLY | UO_CREAT | UO_EXCL | UO_TRUNC, 0600);
```

**INT13-C-EX2:** If the right-side operand to a shift operator is known at compile time, it is acceptable for the value to be represented with a signed type provided it is positive.

```c
#define SHIFT 24
foo = 15u >> SHIFT;
```

Risk Assessment

Performing bitwise operations on signed numbers can lead to buffer overflows and the execution of arbitrary code by an attacker in some cases, unexpected or implementation-defined behavior in others.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT13-C</td>
<td>High</td>
<td>Unlikely</td>
<td>Medium</td>
<td>P6</td>
<td>L2</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>Astére</td>
<td>19.04</td>
<td>bitop-type</td>
<td>Fully checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-INT13</td>
<td></td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>LANG.TYPE.IOT</td>
<td>Inappropriate operand type</td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td></td>
<td></td>
<td>Can detect violations of this rule. In particular, it flags bitwise operations that involved variables not declared with unsigned type</td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.INT13</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>MISRA.BITS.NOT_UNSIGNED</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-INT13-a</td>
<td>Bitwise operators shall only be applied to operands of unsigned underlying type</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT C. Rec. INT13-C</td>
<td>Checks for bitwise operation on negative value (rec. fully covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>4532, 4533, 4534, 4543, 4544</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>bitop-type</td>
<td>Fully checked</td>
</tr>
<tr>
<td>SonarQube C/C++ Plugin</td>
<td>3.11</td>
<td>S874</td>
<td></td>
</tr>
<tr>
<td>Splint</td>
<td>3.1.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the [CERT website](https://www.certi.org).

Related Guidelines

<table>
<thead>
<tr>
<th>SEI CERT C++ Coding Standard</th>
<th>VOID INT13-CPP. Use bitwise operators only on unsigned operands</th>
</tr>
</thead>
</table>
ISO/IEC TR 24772:2013
Bit Representations [STR]
Arithmetic Wrap-around Error [FIF]
Sign Extension Error [XZI]

MITRE CWE
CWE-682, Incorrect calculation

Bibliography

[Dowd 2006] Chapter 6, "C Language Issues"
[C99 Rationale 2003] Subclause 6.5.7, "Bitwise Shift Operators"