INT15-C. Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types

Few programmers consider the issues around formatted I/O and type definitions. A programmer-defined integer type might be any type supported by the implementation, even a type larger than unsigned long long. For example, given an implementation that supports 128-bit unsigned integers and provides a uint_fast128_t type, a programmer can define the following type:

```
typedef uint_fast128_t mytypedef_t;
```

Furthermore, the definition of programmer-defined types may change, which creates a problem when these types are used with formatted output functions, such as printf(), and formatted input functions, such as scanf(). (See FIO47-C. Use valid format strings.)

The C intmax_t and uintmax_t types can represent any value representable by any other integer types of the same signedness. (See INT00-C. Understand the data model used by your implementation(s).) This capability allows conversion between programmer-defined integer types (of the same signedness) and intmax_t and uintmax_t:

```
mytypedef_t x;
uintmax_t temp;

/* ... */
if (temp <= MYTYPEDEF_MAX) {
  x = temp;
}
```

Formatted I/O functions can be used to input and output greatest-width integer typed values. The j length modifier in a format string indicates that the following d, i, o, u, x, or X conversion specifier will apply to an argument with type intmax_t or uintmax_t. C also specifies the z length modifier for use with arguments of type size_t and the c length modifier for arguments of type ptrdiff_t.

In addition to programmer-defined types, there is no requirement that an implementation provide format-length modifiers for implementation-defined integer types. For example, a machine with an implementation-defined 48-bit integer type may not provide format-length modifiers for the type. Such a machine still must have a 64-bit long long, with intmax_t being at least that large.

Noncompliant Code Example (printf())

This noncompliant code example prints the value of x as an unsigned long long value even though the value is of a programmer-defined integer type:

```
#include <stdio.h>

mytypedef_t x;

/* ... */
printf("%llu", (unsigned long long) x);
```

There is no guarantee that this code prints the correct value of x, as x may be too large to represent as an unsigned long long.

Compliant Solution (printf())

The C intmax_t and uintmax_t can be safely used to perform formatted I/O with programmer-defined integer types by converting signed programmer-defined integer types to intmax_t and unsigned programmer-defined integer types to uintmax_t, then outputting these values using the j length modifier. Similarly, programmer-defined integer types can be input to variables of intmax_t or uintmax_t (whichever matches the signedness of the programmer-defined integer type) and then converted to programmer-defined integer types using appropriate range checks.

This compliant solution guarantees that the correct value of x is printed, regardless of its length, provided that mytypedef_t is an unsigned type:
Compliant Solution (Microsoft `printf()`)  
Visual Studio 2012 and earlier versions do not support the standard `j` length modifier and do not have a nonstandard analog. Consequently, the programmer must hard code the knowledge that `intmax_t` is `int64_t` and `uintmax_t` is `uint64_t` for Microsoft Visual Studio versions.

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

mytypedef_t x;
/* ... */

#ifdef _MSC_VER
   printf("%llu", (uintmax_t) x);
#else
   printf("%ju", (uintmax_t) x);
#endif
```

A feature request has been submitted to Microsoft to add support for the `j` length modifier to a future release of Microsoft Visual Studio.

Noncompliant Code Example (`scanf()`)  
This noncompliant code example reads an `unsigned long long` value from standard input and stores the result in `x`, which is of a programmer-defined integer type:

```c
#include <stdio.h>

mytypedef_t x;
/* ... */

if (scanf("%llu", &x) != 1) {
   /* Handle error */
}
```

This noncompliant code example can result in a buffer overflow if the size of `mytypedef_t` is smaller than `unsigned long long`, or it might result in an incorrect value if the size of `mytypedef_t` is larger than `unsigned long long`. Moreover, `scanf()` lacks the error checking capabilities of alternative conversion routines, such as `strtol()`. For more information, see INT06-C. Use `strtol()` or a related function to convert a string token to an integer.

Compliant Solution (`strtoumax()`)  
This compliant solution guarantees that a correct value in the range of `mytypedef_t` is read, or an error condition is detected, assuming the value of `MYTYPEDEF_MAX` is correct as the largest value representable by `mytypedef_t`. The `strtoumax()` function is used instead of `scanf()` as it provides enhanced error checking functionality. The `fgets()` function is used to read input from `stdin`. 
Risk Assessment

Failure to use an appropriate conversion specifier when inputting or outputting programmer-defined integer types can result in buffer overflow and lost or misinterpreted data.

<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>INT15-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>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-INT15</td>
<td>Can catch violations of this rule by scanning the printf() and scanf() family of functions. For each such function, any variable that corresponds to a %d qualifier (or any qualifier besides %j) and that is not one of the built-in types (char, short, int, long, long long) indicates a violation of this rule. To catch violations, ROSE would also have to recognize derived types in expressions, such as size_t.</td>
</tr>
<tr>
<td>Compass /ROSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>586 S</td>
<td>Enhanced Enforcement</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-INT15-a</td>
<td>The basic types of char, int, short, long, float and double should not be used, but specific-length equivalents should be typedef'd</td>
</tr>
</tbody>
</table>
Related Vulnerabilities

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

Related Guidelines

<table>
<thead>
<tr>
<th>SEI CERT C++ Coding Standard</th>
<th>VOID INT15-CPP. Use intmax_t or uintmax_t for formatted IO on programmer-defined integer types</th>
</tr>
</thead>
<tbody>
<tr>
<td>MITRE CWE</td>
<td>CWE-681, Incorrect conversion between numeric types</td>
</tr>
</tbody>
</table>

Bibliography

[Saks 2007c] Standard C's Pointer Difference Type