ISO/IEC JTC 1/SC 22/WG14/N1394 - Analyzability

15 August 2009

Thomas Plum, tplum@plumhall.com

Robert Seacord, rcs@cert.org

Introduction

Please refer to WG14/N1350 [N1350] for prior context which will not be re-stated here. At Markham, the Bounds-Checking Library (BCL) was adopted as a separate conditionally-normative annex. This paper addresses the issues not included in that BCL annex. The minutes [N1380] record the discussion and the votes:

[Begin quotation from N1380]

4.6 Proposal for an Annex to C1X (N1350) (Plum)

N1350 is a proposal to add an annex to C1X that addresses issues that have come out of the work of SC22/WG23, Programming Language Vulnerabilities. There are four proposals in the paper:
1. Add two definitions to the main body of C1X (trap representation, to perform a trap).
2. Add an informative Annex to C1X for solutions to various C vulnerabilities.
3. Add four definitions to the new annex
- to perform a trap
- out-of-bounds-store
- bounded undefined behavior
- critical undefined behavior
4. Add the contents of TR24731-1 to the new annex.

[Discussion]
Proposal #4
Focus initially is #4, as an informative new annex. Should we do at least this much? Tom would like to see this as a conditionally normative annex, i.e. if you support this, do it this way, however support is optional. There has been some NB feedback to not impose a lot of new requirements in C1X. David K believes it should be merged in-line.

TR 24731-1 was put together as an approach to retrofit existing code, and considered by some as not the optimal approach for new code. Hence we started TR24734-2, which has been through DTR ballot, and for which we are waiting final text. If we put too much in the Standard. we dilute the entire Standard. Complex is a likely example.

Lots of discussion, but general support for a conditionally]normative annex. Tom considers such a friendly amendment.

Straw Poll: Support for adding TR24731-1 as a conditionally normative Annex to C1X.
Yes - 16
No - 2
Abs - 0
DECISION: Add TR 24731-1 as a conditionally normative Annex to C1X.

Proposal #1
Adds the following new definitions for terms that are not defined in the definitions, or for terms that are defined in-line in the Standard, but not in the definitions.

trap representation

to perform a trap

Do these definitions conflict with the ISO Vocabulary standard?
ACTION: Tom Plum to check that the proposed definitions in N1350 do not conflict with the ISO Vocabulary standard. [TP - later confirmed, no conflict]

We do not want to constrain what implementations do when a trap occurs.

Straw Poll: Adopt definitions for 'trap representation' and 'to perform a trap' in N1350 to C1X.
Yes - 15
No - 1
Abs - 3
DECISION: Adopt definitions for 'trap representation' and 'to perform a trap' in Proposal #1, N1350 to C1X.

[Tues - PM]
Proposal #2: Add an informative annex to C1x for solutions to various C vulnerabilities (called "Annex K" in this paper), with a tentative title such as "Analyzability".

Proposal #3: Provides a number of definitions, and a table of undefined behavior reclassified (elevated) as critical undefined behavior:

N.1 to perform a trap
As defined at 3.17.5, "to perform a trap" means to interrupt execution of the program such that no further operations are performed. However, in this Annex K, this interruption is permitted to take place by invoking a runtime-constraint handler. Any such semantics are implementation-defined.

N.2 out-of-bounds store
an (attempted) access (3.4.1) which, at run-time, for a given computational state, would modify one or more bytes (or for an object declared volatile, would fetch one or more bytes) that lie outside the bounds permitted by the standard

N.3 bounded undefined behavior
behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements, except that the behavior shall not perform an out-of-bounds store, and that all values produced or stored are indeterminate values.
NOTE The behavior might perform a trap.

N.4 critical undefined behavior
behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements
NOTE the behavior might perform an out-of-bounds store, or might perform a trap

All undefined behaviors defined in clauses 1 through 7 of this International Standard are (in this Annex K) restricted to the semantics of bounded undefined behavior, except for the following critical undefined behaviors:

6.2.4 An object is referred to outside of its lifetime
6.3.2.1 An lvalue does not designate an object when evaluated
6.5.3.2 The operand of the unary * operator has an invalid value
6.5.6 Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that points just beyond the array object and is used as the operand of a unary * operator that is evaluated
7.20.3 The value of a pointer that refers to space deallocated by a call to the free or realloc function is used
7.21.1, 7.24.4 A string or wide string utility function is instructed to access an array beyond the end of an object

PJ: Trouble enumerating between things that are bad with everything that is really bad. Believes there should be additional Undefined Behavior on this list.
Clark: This is really a basis for work to improve the reliability of software.
Tom: This work came from the work OWGV / WG23, and is directly applicable to C. The belief is that WG14 is the correct group to vet this material.
PJ: Does not believe it belongs in the C Standard, even as an informative annex. It's an interesting work in progress.

Straw Poll: Adopt proposals #2 and #3 in N1350 to C1X.
Yes - 9
No - 10
Abs - 3
No consensus to add this to the Standard at this time

Tom plans to bring this to the next meeting, and try to convince the committee to adopt it. There are other forms this work can take, TR, position paper, but additional work is needed. Randy: This is still in a fairly rough state, but in some sense there is still a lot of work to be done. The problem should continue to be addressed.

[End of quotation from N1380]

Avoiding Vulnerabilities

We propose to add a C1x annex defining compiler restrictions to assist with implementation of solutions to the problems defined in N1350.

The actual title for the annex might be one of the last decisions to be made: let's first agree on the contents, and then we can pick the title. The word "vulnerability" is a negative word that defines the problem; presumably we want a positive word that characterizes some aspects of the solution. For now,

we suggest "Analyzability", because there seemed to be general agreement at Milpitas that the ability to analyze the C source was essential to all the audiences listed by WG23. An implementation is operating in "analyzable mode" if it implements the features described in this Analyzability Annex. For now, we propose that the annex be conditionally normative. Other categories are possible, but let's first determine the contents and then we can consider the appropriate category.

There was some discussion not recorded in the Markham minutes, but we recall some suggestions from Tydeman and Plauger. Tydeman observed that the standard explicitly calls out bad arguments to library functions as an undefined behavior. That is correct, and even if those library functions are not implemented in C, those arguments could cause an out-of-bounds store. Plauger observed that calling an incompatible-type function via pointer-to-function might cause out-of-bounds store as a direct result. We agree with both observations, and have added the corresponding undefined-behavior citations to the table below.

PROPOSAL #1:

Add an conditionally normative annex to C1x for solutions to various C vulnerabilities, with a tentative title such as "Analyzability".

[End of Proposal #1]

PROPOSAL #2:

Provide the following definitions in the Analyzability Annex:

N.1 *to perform a trap*
As defined at 3.17.5, "to perform a trap" means to interrupt execution of the program such that no further operations are performed. However, in this Annex, this interruption is permitted to take place by invoking a runtime-constraint handler. Any such semantics are implementation-defined.

N.2 *out-of-bounds store*
an (attempted) access (3.4.1) which, at run-time, for a given computational state, would modify one or more bytes (or for an object declared volatile, would fetch one or more bytes) that lie outside the bounds permitted by the standard

N.3 *bounded undefined behavior*
behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements, except that the behavior shall not perform an out-of-bounds store, and that all values produced or stored are indeterminate values.
NOTE The behavior might perform a trap.

N.4 *critical undefined behavior*
behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements
NOTE the behavior might perform an out-of-bounds store, or might perform a trap

All undefined behaviors defined in clauses 1 through 7 of this International Standard are (in this Annex) restricted to the semantics of bounded undefined behavior, except for the following critical undefined behaviors:

6.2.4	An object is referred to outside of its lifetime
6.3.2.1	An lvalue does not designate an object when evaluated
6.5.3.2	The operand of the unary * operator has an invalid value
6.5.6	Addition or subtraction of a pointer into, or just beyond, an array object and an integer type produces a result that points just beyond the array object and is used as the operand of a unary * operator that is evaluated
7.20.3	The value of a pointer that refers to space deallocated by a call to the free or realloc function is used
7.21.1, 7.24.4	A string or wide string utility function is instructed to access an array beyond the end of an object
6.3.2.3	A pointer is used to call a function whose type is not compatible with the pointed-to type
7.1.4	An argument to a library function has an invalid value or a type not expected by a function with variable number of arguments

[End of Proposal #2]

References

[N1350] “Analyzability", ISO/IEC JTC 1/SC 22/WG14 N1350. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1350.htm

[N1380] "ISO/JTC1/SC22/WG14 AND INCITS PL22.11 SEPTEMBER 2008 MEETING MINUTES (DRAFT)", ISO/IEC JTC 1/SC 22/WG14 N1380. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1380.pdf