ISO/IEC JTC 1/SC 22/WG 23 N 0279

Prototype table summarizing vulnerabilities

Date	10 September 2010
Contributed by	Jim Moore
Original file name	
Notes	Responds to Action Item 14-04

[This is draft 2—incorporating some suggestions from Clive Pygott—of a table that would summarize vulnerabilities. Only a few vulnerabilities are treated in this prototype.]

Annex <whatever>
(Informative)
Summary of Language Vulnerabilities

[The table would actually be produced in landscape format. I've left this prototype in portrait format for ease of review.]

Code	Vulnerability	General	C	Ada
AJN	Choice of filenames and external identifiers	**Risk** • External names have to be compatible with the naming system of all languages and OS **Mitigation** • Use only portable file names		
BJL	Namespace issues			
BQF	Unspecified behaviour		**Risk** • 54 instances listed in Annex J.1 of standard **Mitigation** • Avoid them	**Risk** • Listed by index entries for "unspecified" and "bounded error" in standard **Mitigation** • Avoid them
BRS	Obscure language features		**Risk** • Library routines • Semantics of `goto` **Mitigation** • Understand unfamiliar library routines before using • Avoid `goto`	**Risk** • Semantics of tasking and exceptions **Mitigation** • Use `pragma Restrictions` except for qualified programmers
EWF	Undefined behaviour		**Risk** • 191 instances listed in Annex J.2 of standard **Mitigation** • Avoid them	**Risk** • Listed by index entries for "erroneous execution" in standard **Mitigation** • Avoid them

Code	Vulnerability	General	C	Ada
FAB	Implementation-defined behaviour		**Risk** • 112 instances listed in Annex J.3 of standard **Mitigation** • Document instances	**Risk** • Listed in Annex M of standard **Mitigation** • Avoid them
IHN	Type system		**Risk** • Implicit conversion may lead to unexpected results **Mitigation** • Pay attention to the rules for conversion • Use explicit casts	**Risk** • None (See Note 1) aside from computed or input values falling outside a range check **Mitigation** • Provide an exception handler for Constraint_Error
MEM	Deprecated language features		**Risk** • `gets` • Backword compatibility options of compilers **Mitigation** • Avoid them	**Risk** • Documented in Annex J of standard **Mitigation** • Avoid them
NAI	Choice of clear names	**Risk** • Some characters (e.g "I" and "1") look alike **Mitigation** • Avoid differentiation using characters that are visually confused • Apply a consistent project style guide	**Risk** • Some compilers only pay attention to the beginning of the name • Two underscores in a row look like a single underscore **Mitigation** • Use short names that are distinguishable in the first few characters • Don't use two underscores together	
NMP	Pre-processor directives		**Risk** • Function-like macros look like functions but have different semantics	

Code	Vulnerability	General	C	Ada
			Mitigation • Prefer inline functions to function-like macros • Fully parenthesize macro arguments and body • Avoid embedding directives and using side-effects in a function-like macro	
STR	Bit representation		**Risk** • Bit operations are permitted even where the standard does not specify bit representation **Mitigation** • Use bit operations only on unsigned types • Pay attention to endian; it may be different internal and external to the machine • Avoid shifts larger than the variable's size	
XYR	Unused variables	**Risk** • Unused variables may indicate a design or implementation flow **Mitigation** • Resolve all compiler warnings	**Mitigation** • If using GCC, use the "unused" attribute for intentionally unused variables	
YOW	Identifier name reuse	**Risk** • Block-structured languages permit names in an inner scope to hide the same name in an enclosing scope. Deleting the inner declaration may lead to unexpected results.	**Risk** • Some compilers only pay attention to the beginning of the name	

Code	Vulnerability	General	C	Ada
		Mitigation • Apply naming conventions that avoid name reuse	**Mitigation** • Use short names that are distinguishable in the first few characters	

Notes:

1. In Ada, the strong typing system prevents the occurrence of many of the vulnerabilities. The language does provide capabilities for circumventing the type system. However, the use of those capabilities is not typical, is clearly marked, and can be disabled with `pragma restrictions`.