ISO/IEC/JTC 1/SC 22/WG 23 N0853

17 December 2018

24772-1 DIS Collated comments with proposed edits

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CA 001 | |  | | 06.02.5 | |  | | ed | | "Avoid explicit type conversion of data values except when there is no alternative. Document such occurrences so that the justification is made available to maintainers."  This could be read as encouraging implicit type conversions over explicit ones, which I believe is the opposite of the intent. Further, while I can possibly imagine it might make sense to document implicit conversions, I think the explicit conversion mostly reduces or eliminates the need to document the conversions. There is no mention in the previous bullet about documenting implicit conversions, where the need is greater. | | Suggest removing "explicit" from this bullet, i.e. "Avoid [explicit] type conversion of data values except when there is no alternative. Document such occurrences so that the justification is made available to maintainers." | | Reject.  Implicit conversions are covered by the preceding paragraph to the one referenced in 6.2.5. | |
| CA 002 | |  | | 06.03.3 | |  | | ed | | "Problems can arise when programmers mix their techniques to reference the bits or output the bits. Problems can arise when programmers mix arithmetic and logical operations to reference the bits or output the bits. " There appears to be two versions of this sentence. Are they both needed? | | Delete one of the two sentences if both are not needed. Otherwise, the second sentence should probably begin with "Problems can {also} arise ...." | | Accept.  Combined into  “Problems can arise when programmers mix their techniques (e.g. arithmetic and logical operations) to reference the bits or output the bit, since storage ordering of the bits may not be what the programmer expects” | |
| PL 003 | |  | | 06.04 | | 06.4.5 | | te | | Last subpoint: should be completed by "or use a suitable compensated summation algorithm" to read | | When adding (or subtracting) sequences of numbers, sort and add (or subtract) them from smallest to largest in absolute value or use a suitable compensated summation algorithm to avoid loss of precision. | | Accepted. | |
| CA 004 | |  | | 06.08.3 | |  | | ed | | “When an array has been allocated storage on the stack{,} an out-of-bounds write access may modify internal runtime housekeeping information (for example, a function's return address) which might change a program’s control flow” | | Add missing comma | | Accept.ed | |
| CA 005 | |  | | 06.14.06 | |  | | te | | "Language specifiers should design generics in such a way that any attempt to instantiate a generic with constructs that do not provide the required capabilities results in a compile-time error." | | This seems completely unrelated to the vulnerability, nothing to do with free, but does sound like it could be applied to a different vulnerability relating to generics and templates. Is this out of place? Perhaps a cut and paste error? | | Accepted | |
| CA 006 | |  | | 06.17.04 | |  | | te | | "Languages that treat letter case as significant. Some languages do not differentiate between names with differing case, while others do." This suggests that the problem is with languages that treat case as significant, but the description in 6.17.1, in the 4th paragraph states; “There is also an issue where identifiers appear distinct to a human but identical to the computer” implies the problem is with languages where case is insignificant. In reality, I think confusion can come from either type of language. But mostly only in cases of languages where variables do not need to be declared. | | Suggest replacing the 4th bullet with a more critical issue;  “Languages that allow variables to be implicitly declared.”  An example that comes to mind here is BASIC, where, at least in earlier versions of the language, one could misspell the name of a variable and implicitly get a second variable, which might not be noticed by the programmer. | | Accept in principle.  Added to 6.17.5, the bullet:  “In languages with optional declarations of variables, always use explicit declarations of the variables to assist compiler checking.” | |
| CA 007 | |  | | 06.22.03 | |  | | te | | This vulnerability appears to cover elaboration errors? That is another source of such an error that is not described here, where a global variable can be used before initialization, even though in the code, the variable appears to be initialized. I do see it mentioned in 6.22.5. | | This can be a very dangerous problem as the code looks correct. I think it would be helpful to describe this possibility better in 6.22.1 and 6.22.3 | | Rejected.  This is covered in 6.22.1 paragraph 3. Elaboration is mentioned but not explicitly called out because this addresses the vulnerability in a language-independent way. | |
| CA 008 | |  | | 06.22.03 | |  | | ed | | "There is a special case [of]{for} pointers or access types." | | Replace “of” with “for”, or perhaps “associated with” | | Accepted. | |
| CA 009 | | 4th bullet | | 06.22.05 | |  | | te | | There is a recommendation to initializing variables during elaboration, but that is precisely what causes this problem in most languages such as C++. The recommendation to initialize during elaboration only is good advice for languages that can perform safe/orderly elaboration based on dependencies, so there should be clarification. A general recommendation for all languages that allow global declarations would be to avoid creating global objects. For example, dont create a singleton object if all that is needed is function calls that generate a result based on the input parameters. For languages that do not support safe elaboration, the recommendation should be to NOT initialize the object during elaboration, but instead either defer creation of the object until after elaboration, or protect access to the object by setting a flag indicating if the object has been initialized or not. | | Clariify the advice is only recomnmended if the language supports safe elaboration, otherwise either try to defer creation of the object until after elaboration (such via access using a pointer), or by putting guards around the object to ensure that the first access either initializes the object, or indicates failure if the object is not yet initialized. | | Accepted in principle.  Corrected by removing “elaboration” since it is too language-specific, and using the term “declaration”. The second point is covered in other bullets. | |
| CA 010 | |  | | 06.23.01 | |  | | ed | | "Each language provides rules of precedence and associativity, for each expression that operands bind to which operators. T" | | I am unable to parse this sentence. Please reword. | | Accepted.  Replaced sentence with  “Each language provides rules of precedence and associativity that determine for each expression which operands bind to which operators.” | |
| CA 011 | | 6th bullet | | 06.25.05 | |  | | ed | | The 6th bullet says avoid null statements, but I think the intent there (expressions that do nothing or are not executed, for instance) is getting confused with language defined null statements, which is actually recommended in 6.25.6, 1st bullet. | | Clarify that the 6th bullet is not referring to language defined null statements, but rather expressions or statements that do not have an effect, or something along those lines. | | Accepted  Removed the parenthetical text | |
| CA 012 | |  | | 06.25.06 | |  | | ed | | "Languages should consider providing warnings for statements that are unlikely to be right such as statements [without]{with} side effects. " | | The logic of the sentence is backwards. “Without” should be replaced with “with”. | | Accepted.  Removed the first bullet that was subject of the comment. | |
| CA 013 | |  | | 06.26.03 | |  | | te | | Another mechanism of failure, that could be mentioned is related to code bloat. A significant portion of dead code might effect the performance of the application (when loading the executable into memory, or perhaps cause linking problems by using up available code space on a tightly constrained embedded system. | |  | | Rejected.  The topic seems unrelated to vulnerabilities. | |
| CA 014 | |  | | 06.26.05 | |  | | ed | | "When a developer identifies code that is dead because a conditional [consistently]{statically} evaluates " | | Suggest replacing “consistently” with “statically” | | Rejected.  It is intentional that dynamic values are included in the case analysed. | |
| CA 015 | |  | | 06.34.04 | |  | | te | | Probably variadic functions should be given special mention, such as printf(), where a format string is passed as an argument, and the logic of library, rather than the compiler, assumes that the number of parameters passed matches the format string. A recommendation for such languages that support variadic functions, could be to use altnernate calls, such as streaming, where the compiler can validate the correct parameters are passed, rather than relying on runtime functions. | | Suggest mentioning variadic functions, and recommend avoiding their use if possible. | | Accepted.  Added a sentence to cover the mechanism of failure. The mitigation was already covered. | |
| CA 016 | |  | | 06.39.04 | |  | | ed | | "Languages {that} reclaim memory under programmer control can exhibit heap fragmentation and memory leaks." | | Add “that” | | Accepted. | |
| CA 017 | |  | | 06.39.06 | |  | | Ed | | The word "pragma" is wrong font and size | |  | | Accepted. | |
| CA 018 | |  | | 06.40.03 | |  | | Ed | | " about the types it can legally be instantiated with." Avoid ending a sentence with a preposition. | | Suggest replacing with => "about the parameterized types." | | Accepted. | |
| CA 019 | |  | | 06.40.03 | | Paragraph 3 | | ed | | There are two "For Example"s, which looks and reads as though one example was meant to replace the other. | | Remove one of the examples, if only 1 is needed, or preface the second with something like, “As a second example,” | | Accepted.  The second “for example” is removed. | |
| CA 020 | |  | | 06.42.03 | |  | | ed | | "of a class which [re]dispatches to the implementation" I dont think this applies only to redispatching, also to ordinary dispatching | | Suggest generalizing and removing "re" | | Accepted. | |
| CA 021 | |  | | 06.50.06 | |  | | te | | "Provide a mechanism to determine which exceptions might be thrown by a called library routine." It is not clear this is good advice.   Based on experience in other languages (in particular Java and C++), exception "signatures" have not generally provided the value expected for them. Here is an example paper describing some of the issues in the context of Java:    <http://literatejava.com/exceptions/checked-exceptions-javas-biggest-mistake/> <<http://literatejava.com/exceptions/checked-exceptions-javas-biggest-mistake/>>  Here is a discussion of the problems with C++ exception specifications: <http://www.gotw.ca/publications/mill22.htm> <<http://www.gotw.ca/publications/mill22.htm>> | | Consider whether this sentence should be removed or not. On the other hand, this is only advice, which the reader can ignore. There may be some who feel this is still good advice. Or perhaps clarification would be helpful to mention possible pitfalls of following this advice. | | Accepted.  Bullet deleted. | |
| PL 022 | |  | | Gereneral | |  | | ge | | the document even though claiming to be language independent mostly depends on how Ada/C languages are implemented and mostly describes just them. Paper should either stay totally language independant and describe general concepts like floating point arithmetics problems etc. or only papers for specific languages should remain. | | Leave only totally language independent topics in the document. | | Accept in principal.  The vulnerabilities described apply to most programming languages in use today. Some terminology is needed to describe each vulnerability. To the extent possible, the authors tried to use language-independent terminology. The authors would appreciate any concrete suggestions for better language-independent terminology for a future revision. | |

ISO\_IEC PDTR 24772-1 - JTC001-SC22-N5310\_PKN.doc: Collation successful

ISO\_IEC PDTR 24772-1 - JTC001-SC22-N5310\_SCC.doc: Collation successful

Collation of files was successful. Number of collated files: 2

SELECTED (number of files): 2

PASSED TEST (number of files conformed to CCT table model): 2

FAILED TEST (number of files conformed to CCT table model): 0

CCT - Version 2018.2