ISO/IEC JTC 1/SC 22/OWGV N 0206

Meeting #11 markup of draft language-specific annex for Fortran

Date	15 July 2009
Contributed by	Dan Nagle
Original file name	
Notes	Markup of N0198

To: WG23
Subject: Draft Fortran Annex
From: Dan Nagle
Date: 2009 June 4

Draft of the Fortran Annex of the WG23 TR 24772

This Annex provides Fortran-specific advice
for the items in clause 6, specifically
using Annex F from n0191.

DRAFT

Language Specific Vulnerability Outline

<appendix>. This Annex provides Fortran-specific advice
for the items in clause 6. Each vulnerability is addressed,
even if only to indicate that the vulnerability does not apply
to Fortran.

<appendix>.1 Identification of standards
ISO/IEC 1539-1 (2010) "Fortran 2008"

Comment [JWM1]: It might be wise to make the point that the legacy features that remain in the standard in deprecated status should be avoided.

<appendix>.2 General Terminology
The Fortran standard specifies the forms of Fortran programs (source code)
may take, and the rules for interpreting them. It also specifies
the form of input and output files. A processor is a combination
of a computing system and the mechanism by which programs are transformed
for use on that computing system. The standard does not describe
the processor, except that, if the program conforms to the standard
then the processor shall interpret the program according to the standard.

A requirement expressed in ISO/IEC 1539-1 is a requirement
on the program, not the processor, unless explicitly stated otherwise.
~~The processor is limited in its required ability to detect errors to those errors that can be found by reference to the numbered syntax rules~~

~~and constraints of the standard~~The processor is required to provide a mode that enforces adherence to the standard.

A behavior not completely specified by ISO/IEC 1539-1
is said to be processor dependent.

Some features from earlier revisions of ISO/IEC 1539-1 are
considered
redundant and largely unused, and are designated decremental
features. The two categories of decremental features are
deleted features, which are no longer a part of the standard,
and obsolescent features, which are part
of the standard, but whose use is discouraged.
There is a modern equivalent for every decremental feature
that is considered easier to use and more clear in its meaning.

A Fortran processor optionally cooperates
with one or more companion processors, that may be compilers
of other languages. A processor is its own coprocessor,
additional coprocessors may be compilers of other languages.
The only requirement is that the other languages allow
their data and procedures to be described in terms of C.
The actions of routines written in a language other than Fortran
are not subject to the rules of Fortran.

%%%%%

Fortran.3.1 Obscure Language Features [BRS]

Fortran.3.1.0 Status and history
Original draft - DLN

Fortran.3.1.1 Language-specific terminology

Fortran is an old language and its standard has been through
many revisions. Several features of older revisions are considered
decremental (either deleted or obsolescent). Also, some early
revisions
were especially simple languages, and did not include all the
functionality
that applications programmers wanted. Thus, processor vendors added
extensions to supply the missing functionality. These extensions
may or may not be well known by modern programmers.

For all new code, ~~the non-obsolescent features of the latest revision of the standard should be used.~~decremental features of the standard should not be used.

Fortran.3.1.2 Description of application vulnerability

Not all programmers are familiar with code written to decades-old

standards, nor are they familiar with extensions, once common, used to add missing features to archaic Fortran. The use of these features is error prone and obscure. Some have unexpected changes of state that are likely to surprise modern programmers.

[Need to discuss the examples: common blocks, initialized variable, etc.]

Fortran.3.1.3 Mechanism of failure

An original programmer decades ago may have understood use of a feature whose use was common at the time, but the entirety of effects of some archaic features may not be known to modern programmers. These effects may produce semantic results not in accord with the modern programmer's intentions. They may be beyond the capability of modern review.

[Need to discuss the examples: common blocks, initialized variable, etc.]

Fortran.3.1.4 Avoiding the vulnerability or mitigating its effects in Fortran

This vulnerability can be avoided or mitigated in Fortran in the following ways:

Use the processor to detect and identify decremental features.
Use the processor to detect and identify extensions.
All decremental features have modern counterparts that are safer, easier to understand, and more parallel to the semantics of other languages.
Thus, the modern alternative should be preferred.

Avoid the use of decremental features.
Avoid the use of processor extensions, including processor-defined intrinsic procedures.
The entire definition of one common block should be entirely within one statement; one statement should define at most one common block.
Be aware that ~~an initial value~~the initialization of a variable implies static storage.

Fortran.3.1.5 Implications for standardization in Fortran

Fortran should continue to identify and ~~depreciate~~ deprecate features whose use is problematic and where there is a safer and more clear alternative in the modern revisions of the language.

Language processors might be required to detect the use of decremental features.

[Might add explicit recommendations regarding definition of common blocks and initialization of storage.]

Fortran.3.1.6 Bibliography

%%%%%

Fortran.3.2 Unspecified Behaviour [BQF]

Fortran.3.2.0 Status and history
Original draft - DLN

Fortran.3.2.1 Language-specific terminology

The use of any form or relation not specified by the Fortran standard
has unspecified effects. The use of any form or relationship given a meaning not specified by the standard has unspecified effects.

The use of any extension has unspecified effects, and such use is not necessarily portable to other processors, nor does it necessarily agree with what the programmer intended.

[It would be good to explain what an instrinsic is.]

Fortran.3.2.2 Description of application vulnerability

A Fortran processor is unconstrained unless the program
uses only those forms and relations specified by the Fortran standard,
and gives them the meaning described therein.

[Might want to explain about intrinsics.]

Fortran.3.2.3 Mechanism of failure

What a processor does with non-standard code is unpredictable. The behavior of non-standard code can change between processors, or between releases of the same processor. It is entirely unpredictable.

[Might want to explain about intrinsics.]

Fortran.3.2.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use processor options to detect and report use of non-standard features.
Use more than one processor to obtain diagnostics from more than one source.
Do not use intrinsic procedures not described in the standard.

Fortran.3.2.5 Implications for standardization in Fortran

None.

Fortran.3.2.6 Bibliography

%%%%%

Fortran.3.3 Undefined Behaviour [EWF]

[If the Fortran standard does not distinguish between unspecified and undefined behaviour, then this section should simply say that and point back to the previous section.]

Fortran.3.3.0 Status and history
Original draft - DLN

Fortran.3.3.1 Language-specific terminology

The use of any form or relation not specified by the Fortran standard
has unspecified effects. The use of any form or relationship given a meaning not specified by the standard has unspecified effects.

The use of any extension has unspecified effects, because such use is not portable to other processors.

Behavior that is specifically undefined is described in Clause 1.1, and in Annex A.1 of ISO/IEC 1539-1 (2010).

Fortran.3.3.2 Description of application vulnerability

A Fortran processor is unconstrained unless the program
uses only those forms and relations specified by the Fortran standard,
and gives them the meaning described therein.

Fortran.3.3.3 Mechanism of failure

What a processor does with non-standard code is unpredictable.
The behavior of non-standard code can change between processors, or between releases of the same processor. This is unpredictable.

Fortran.3.3.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use processor options to detect and report use of non-standard features.
Use more than one processor to obtain diagnostics from more than one source.

Fortran.3.3.5 Implications for standardization in Fortran

Fortran.3.3.6 Bibliography

%%%%%

Fortran.3.4 Implementation-defined Behaviour [FAB]

Fortran.3.4.0 Status and history
Original draft - DLN

Fortran.3.4.1 Language-specific terminology

Implementation-defined behavior is called processor dependent behavior
in Fortran. See Annex A.2 of ISO/IEC 1539-1 for a list of processor dependencies.

[Beyond this, it may not be necessary to say anything in this section beyond what is in the body of the TR.]

Fortran.3.4.2 Description of application vulnerability

Different processors may process different processor-dependencies differently. Relying on one behavior is not guaranteed by the Fortran standard.

Fortran.3.4.3 Mechanism of failure

Reliance on one behavior where the standard explicitly allows several is not portable, and is liable to change between releases of a single processor, or between different processors.

Fortran.3.4.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not rely on processor dependencies. See Annex A.2 for a complete list.

[Is it sensible to recommend the use of “inquiry intrinsics”?]

[Perhaps you should include a recommendation to test using different compilers and different options.]

Fortran.3.4.5 Implications for standardization in Fortran

Fortran.3.4.6 Bibliography

%%%%%

Fortran.3.5 Deprecated Language Features [MEM]

Fortran.3.5.0 Status and history
Original draft - DLN

Fortran.3.5.1 Language-specific terminology

Several features of older revisions are considered
decremental (either deleted or obsolescent). Modern synonyms
exist for each decremental feature.

Fortran.3.5.2 Description of application vulnerability

Decremental features are often obscure, and therefore might
not be fully understood by modern programmers. They may have
end cases or side effects that are not clear.

Fortran.3.5.3 Mechanism of failure

Side effects or end cases unknown to a modern programmer may cause
unpredictable behavior.

Fortran.3.5.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Do not use decremental features.
Use the processor to detect and identify decremental features;
then replace them with a modern synonym.
Use processor options to require adherence to the latest standard.

Fortran.3.5.5 Implications for standardization in Fortran

Fortran.3.5.6 Bibliography

%%%%%

Fortran.3.6 Pre-processor Directives [NMP]

Fortran.3.6.0 Status and history
Original draft - DLN

Fortran.3.6.1 Language-specific terminology

No preprocessor is part of ISO/IEC 1539-1 (2010).

Fortran.3.6.2 Description of application vulnerability

N/A

Fortran.3.6.3 Mechanism of failure

N/A

Fortran.3.6.4 Avoiding the vulnerability or mitigating its effects in Fortran

N/A

Fortran.3.6.5 Implications for standardization in Fortran

Fortran.3.6.6 Bibliography

%%%%%

Fortran.3.7 Choice of Clear Names [NAI]

Fortran.3.7.0 Status and history
Original draft - DLN

Fortran.3.7.1 Language-specific terminology

Fortran is a single case language, upper case and lower case must be treated identically by the processor. Fortran has keywords but no reserved words.

Fortran.3.7.2 Description of application vulnerability

Use of names differing only in capitalization, intended to be distinct,
in fact are not distinct. While some processors have options to preserve
case of names, others do not. In any case, using case to distinguish
names directly contradicts the standard and should be shunned.

Use of a keyword as a name may be possible, but is confusing and should be shunned.

[You might want to mention processor dependencies in accepting non-standard character sets in identifier names.]

[This description does not address all of the issues in the NAI description in the body of the standard, e.g. underscores.]

Fortran.3.7.3 Mechanism of failure

The name distinguished by case is not standard, and even if it works with one processor, is not portable.

Using a keyword as a name is confusing.

Fortran.3.7.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not distinguish names by case only.

Do not use keywords as names.

Fortran.3.7.5 Implications for standardization in Fortran

Processors might detect and report the occurrence of names
appearing to differ only in case.

Processors might detect and report the occurrence of names
indistinguishable from keywords.

Fortran.3.7.6 Bibliography

%%%%%

Fortran.3.8 Choice of Filenames and other External Identifiers [AJN]

Fortran.3.8.0 Status and history
Original draft - DLN

Fortran.3.8.1 Language-specific terminology

Filenames appearing in OPEN and INQUIRE statements, and character
variables
in references to the GET_ENVIRONMENT_VARIABLE intrinsic
and the EXECUTE_COMMAND_LINE intrinsic, have trailing blanks
removed.

Fortran considers any filename, environment variable name, or
name specified on an INCLUDE line to be processor-dependent.

Fortran.3.8.2 Description of application vulnerability

[Perhaps there should be a statement saying that Fortran is not much different than the description in the body of the document.]

Fortran.3.8.3 Mechanism of failure

Filenames and environment variable names have trailing blanks
removed before being passed to the operating system. Thus,
two names differing only by trailing blanks
cannot be distinguished.

Parameterize any directory name separators used.

Fortran.3.8.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Do not attempt to distinguish names by trailing blanks.

Fortran.3.8.5 Implications for standardization in Fortran

Fortran.3.8.6 Bibliography

%%%%%

Fortran.3.9 Unused Variable [XYR]

Fortran.3.9.0 Status and history
Original draft - DLN

Fortran.3.9.1 Language-specific terminology

Fortran.3.9.2 Description of application vulnerability

[The explanation of the relationship between Fortran and the body of the TR is missing.]

Fortran.3.9.3 Mechanism of failure

Fortran.3.9.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use IMPLICIT NONE to require explicit declarations.
Use any available processor options to report unused variables.

Do not use common blocks, as the common may legitimately contain names of variables unused in one subprogram.

Use ONLY clauses on USE statements to indicate the names being accessed by use association. Use rename clauses to avoid name collisions.

Fortran.3.9.5 Implications for standardization in Fortran

Fortran might consider requiring processors to have the ability to detect and report the occurrence of unused variables.

Fortran.3.9.6 Bibliography

%%%%%

Fortran.3.10 Identifier Name Reuse [YOW]

Fortran.3.10.0 Status and history
Original draft - DLN

Fortran.3.10.1 Language-specific terminology

Fortran names may contain up to 63 characters, all of which are significant. Thus, a name is either illegal or all its characters are used.

Fortran.3.10.2 Description of application vulnerability

Internal procedures access the names available in their hosts.
Module procedures access the names available in their module.
Blocks access the names available in their host.

Fortran.3.10.3 Mechanism of failure

Fortran.3.10.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not use BLOCKS. Do not use nested declarations in DO CONCURRENT blocks.
Use an ONLY clause on all USE statements. Check names
in nested procedures.

Prefer placing subprograms in modules rather than
as internal procedures. Use different modules for data
and for procedures.

Fortran.3.10.5 Implications for standardization in Fortran

Consider adding a means to control host association.
Consider decrementing BLOCKS and declarations in DO CONCURRENT statements.

Fortran.3.10.6 Bibliography

%%%%%

Fortran.3.11 Type System [IHN]

Fortran.3.11.0 Status and history
Original draft - DLN

Fortran.3.11.1 Language-specific terminology

A type defined by the standard is an intrinsic type.
A type defined by the programmer is a derived type.
Some derived types are defined in standard defined modules.

Fortran.3.11.2 Description of application vulnerability

Fortran promotes operands in expressions from smaller to larger within a numeric type, and among types, from integer to real to complex.

Fortran expressions are evaluated without regard to context;
the type of an expression is converted as needed to the type
of the designator receiving the value.

Fortran.3.11.3 Mechanism of failure

Fortran.3.11.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.11.5 Implications for standardization in Fortran

Consider adding a capability to report, forbid, or control automatic conversions.

Consider adding an inquiry intrinsic to provide the largest integer a real kind is capable of representing exactly.

Fortran.3.11.6 Bibliography

%%%%%

Fortran.3.12 Bit Representations [STR]

Fortran.3.12.0 Status and history
Original draft - DLN

Fortran.3.12.1 Language-specific terminology

A bit representation might be made visible when the same storage location
in a storage sequence has names with different types. This can occur
when a common block has different names in different scopes, or when an equivalence has names of different types for the same location.
Also, the TRANSFER intrinsic copies bits between variables without regard
for their types.

Misuse of the Interoperability with C features of Fortran may result in a violation of the Fortran type system.

Fortran.3.12.2 Description of application vulnerability

Fortran.3.12.3 Mechanism of failure

Fortran.3.12.4 Avoiding the vulnerability or mitigating its effects in Fortran

Always use the same definition for a common block in every scoping unit.
Better, convert common blocks to modules. Use renames to retain previous names as needed.

Keep the same type for all variables in an equivalence set.

Do not use TRANSFER.

Ensure type consistency when passing pointers to coprocessor routines.

Fortran.3.12.5 Implications for standardization in Fortran

Processor might have the ability to detect and report the occurrence of storage locations with more then one type, and report the use of the TRANSFER intrinsic.

Fortran.3.12.6 Bibliography

%%%%%

Fortran.3.13 Floating-point Arithmetic [PLF]

Fortran.3.13.0 Status and history
Original draft - DLN

Fortran.3.13.1 Language-specific terminology

A floating point number is of type REAL. There are several (at least two)
kinds of type real supported by any processor. The type kind values parameterize the precision and range (of exponent) supported.

Concerns over floating point characteristics also apply to complex data and operations.

Possibly distinct type kind values are available via the IEEE intrinsic
modules.

Fortran.3.13.2 Description of application vulnerability

Fortran.3.13.3 Mechanism of failure

Fortran.3.13.4 Avoiding the vulnerability or mitigating its effects in Fortran

Where precise control of floating point operations is required, use the IEEE intrinsic modules.

Use trusted libraries to perform common operations (such as linear algebra, Fourier transforms, minimax problems, and so on).

Fortran.3.13.5 Implications for standardization in Fortran

Processors might be required to have an option to detect and report the occurrence of tests for floating point equality.

Fortran.3.13.6 Bibliography

%%%%%

Fortran.3.14 Enumerator Issues [CCB]

Fortran.3.14.0 Status and history

Fortran.3.14.1 Language-specific terminology

The Fortran enumerator type is designed solely for interoperability with the C enumerator type, and should not be used for other purposes.

Fortran.3.14.2 Description of application vulnerability

Fortran.3.14.3 Mechanism of failure

Fortran.3.14.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.14.5 Implications for standardization in Fortran

Fortran might consider defining a standard enumerator type for uses beyond interoperability with C.

Fortran.3.14.6 Bibliography

%%%%%

Fortran.3.15 Numeric Conversion Errors [FLC]

Fortran.3.15.0 Status and history
Original draft - DLN

Fortran.3.15.1 Language-specific terminology

Fortran.3.15.2 Description of application vulnerability

Fortran.3.15.3 Mechanism of failure

Fortran.3.15.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.15.5 Implications for standardization in Fortran

Fortran might consider adding a requirement that processors have the ability to detect and report conversions that might result in loss of data.

Fortran might consider adding an inquiry intrinsic to report the largest integer a real type can represent exactly.

Fortran.3.15.6 Bibliography

%%%%%

Fortran.3.16 String Termination [CJM]

Fortran.3.16.0 Status and history
Original draft - DLN

Fortran.3.16.1 Language-specific terminology

Fortran has two varieties of character assignment: One has
a truncate or blank fill semantic; the other causes a re-allocation
of the target of the assignment when needed. No string terminator
is used in the standard language.

Fortran.3.16.2 Description of application vulnerability

Fortran.3.16.3 Mechanism of failure

Fortran.3.16.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Fortran.3.16.5 Implications for standardization in Fortran

Fortran.3.16.6 Bibliography

%%%%%

Fortran.3.17 Boundary Beginning Violation [XYX]

Fortran.3.17.0 Status and history
Original draft - DLN

Fortran.3.17.1 Language-specific terminology

Fortran uses the term <rank> for the number of dimensions
of an array.

An array of rank <n> is said to have <n> extents.
Each extent has an upper and a lower bound.
Using subscripts with values outside the bounds is prohibited.
An index is called a subscript in Fortran.

Fortran.3.17.2 Description of application vulnerability

Fortran.3.17.3 Mechanism of failure

Fortran.3.17.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Use whole array operations and intrinsics where possible.
Use inquiry intrinsics to determine upper and lower bounds.
Choose upper and lower bounds that naturally describe the problem.

Use assumed shape arrays when passing array arguments.
Use allocatable, automatic, or fixed shape local arrays.

Fortran.3.17.5 Implications for standardization in Fortran

Fortran.3.17.6 Bibliography

%%%%%

Fortran.3.18 Unchecked Array Indexing [XYZ]

Fortran.3.18.0 Status and history
Original draft - DLN

Fortran.3.18.1 Language-specific terminology

Fortran.3.18.2 Description of application vulnerability

Fortran.3.18.3 Mechanism of failure

Fortran.3.18.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use whole array operations and intrinsics where possible.
Use inquiry intrinsics to determine upper and lower bounds.
Choose upper and lower bounds that naturally describe the problem.
Use assumed shape arrays when passing array arguments.
Use allocatable, automatic, or fixed shape local arrays.

Fortran.3.18.5 Implications for standardization in Fortran

Fortran.3.18.6 Bibliography

%%%%%

Fortran.3.19 Unchecked Array Copying [XYW]

Fortran.3.19.0 Status and history

Fortran.3.19.1 Language-specific terminology

Fortran.3.19.2 Description of application vulnerability

Fortran.3.19.3 Mechanism of failure

Fortran.3.19.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use whole array operations and intrinsics where possible.
Use inquiry intrinsics to determine upper and lower bounds.
Choose upper and lower bounds that naturally describe the problem.
Use assumed shape arrays when passing array arguments.

Use allocatable, automatic, or fixed shape local arrays.

Fortran.3.19.5 Implications for standardization in Fortran

Fortran.3.19.6 Bibliography

%%%%%

Fortran.3.20 Buffer Overflow [XZB]

Fortran.3.20.0 Status and history
Original draft - DLN

Fortran.3.20.1 Language-specific terminology

Fortran.3.20.2 Description of application vulnerability

Fortran.3.20.3 Mechanism of failure

Fortran.3.20.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use whole array operations and intrinsics where possible.
Use inquiry intrinsics to determine upper and lower bounds.
Choose upper and lower bounds that naturally describe the problem.
Use assumed shape arrays when passing array arguments.
Use allocatable, automatic, or fixed shape local arrays.

Fortran.3.20.5 Implications for standardization in Fortran

Fortran.3.20.6 Bibliography

%%%%%

Fortran.3.21 Pointer Casting and Pointer Type Changes [HFC]

Fortran.3.21.0 Status and history
Original draft - DLN

Fortran.3.21.1 Language-specific terminology

Fortran pointers are strongly typed, and may point only to variables named as targets.

Fortran.3.21.2 Description of application vulnerability

Fortran.3.21.3 Mechanism of failure

Fortran.3.21.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.21.5 Implications for standardization in Fortran

Fortran.3.21.6 Bibliography

%%%%%

Fortran.3.22 Pointer Arithmetic [RVG]

Fortran.3.22.0 Status and history
Original draft - DLN

Fortran.3.22.1 Language-specific terminology

Fortran does not allow pointer arithmetic.

Fortran.3.22.2 Description of application vulnerability

Fortran.3.22.3 Mechanism of failure

Fortran.3.22.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.22.5 Implications for standardization in Fortran

Fortran.3.22.6 Bibliography

%%%%%

Fortran.3.23 Null Pointer Dereference [XYH]

Fortran.3.23.0 Status and history
Original draft - DLN

Fortran.3.23.1 Language-specific terminology

Fortran.3.23.2 Description of application vulnerability

Fortran.3.23.3 Mechanism of failure

Fortran.3.23.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use the ALLOCATED and ASSOCIATED intrinsics to guard
against using pointers without targets.

Fortran.3.23.5 Implications for standardization in Fortran

Fortran.3.23.6 Bibliography

%%%%%

Fortran.3.24 Dangling Reference to Heap [XYK]

Fortran.3.24.0 Status and history
Original draft - DLN

Fortran.3.24.1 Language-specific terminology

Fortran.3.24.2 Description of application vulnerability

Fortran.3.24.3 Mechanism of failure

Fortran.3.24.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use assignment statements to give variables initial values
on entry to procedures. Be aware that an initial value
in a declarative statement implies static storage.

Do not apply the SAVE attribute to allocatable local variables.

Fortran.3.24.5 Implications for standardization in Fortran

Fortran.3.24.6 Bibliography

%%%%%

Fortran.3.25 Templates and Generics [SYM]

Fortran.3.25.0 Status and history
Original draft - DLN

Fortran.3.25.1 Language-specific terminology

Fortran does not support templates or generics.

Fortran.3.25.2 Description of application vulnerability

Fortran.3.25.3 Mechanism of failure

Fortran.3.25.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.25.5 Implications for standardization in Fortran

Fortran.3.25.6 Bibliography

%%%%%

Fortran.3.26 Inheritance [RIP]

Fortran.3.26.0 Status and history
Original draft - DLN

Fortran.3.26.1 Language-specific terminology

A type that inherits from another type is said to be a child type
that extends the parent type. Fortran supports single inheritance
only.
Polymorphic variables are limited to pointers, allocatable
variables,
and dummy arguments.

Fortran.3.26.2 Description of application vulnerability

Fortran.3.26.3 Mechanism of failure

Fortran.3.26.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Fortran.3.26.5 Implications for standardization in Fortran

Fortran might consider adding class invariants.

Fortran.3.26.6 Bibliography

%%%%%

Fortran.3.27 Initialization of Variables [LAV]

Fortran.3.27.0 Status and history
Original draft - DLN

Fortran.3.27.1 Language-specific terminology

Supplying an initial value to a variable implies
that static storage will be used, not that the variable
is initialized whenever the scope in which it is declared is
entered.

Fortran.3.27.2 Description of application vulnerability

Fortran.3.27.3 Mechanism of failure

Fortran.3.27.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Use executable statements to supply initial values
to procedure local variables.

Fortran.3.27.5 Implications for standardization in Fortran

Fortran might consider a way to supply an initial value
for a variable every time the scope in which it is declared is
entered.

Fortran.3.27.6 Bibliography

%%%%%

Fortran.3.28 Wrap-around Error [XYY]

Fortran.3.28.0 Status and history
Original draft - DLN

Fortran.3.28.1 Language-specific terminology

Fortran.3.28.2 Description of application vulnerability

Fortran.3.28.3 Mechanism of failure

Fortran.3.28.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not use the same variables for bit operations and for arithmetic; instead, use separate variables and check values upon conversion.

Fortran.3.28.5 Implications for standardization in Fortran

Fortran might consider a separate type for bit operations.

Fortran.3.28.6 Bibliography

%%%%%

Fortran.3.29 Sign Extension Error [XZI]

Fortran.3.29.0 Status and history
Original draft - DLN

Fortran.3.29.1 Language-specific terminology

Fortran does not have unsigned data types.

Fortran.3.29.2 Description of application vulnerability

Fortran.3.29.3 Mechanism of failure

Fortran.3.29.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.29.5 Implications for standardization in Fortran

Fortran.3.29.6 Bibliography

%%%%%

Fortran.3.30 Operator Precedence/Order of Evaluation [JCW]

Fortran.3.30.0 Status and history
Original draft - DLN

Fortran.3.30.1 Language-specific terminology

Assignment is not an operator in Fortran.
Bit operations are intrinsic functions, so precedence is clear.

Fortran.3.30.2 Description of application vulnerability

Fortran.3.30.3 Mechanism of failure

Fortran.3.30.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.30.5 Implications for standardization in Fortran

Fortran.3.30.6 Bibliography

%%%%%

Fortran.3.31 Side-effects and Order of Evaluation [SAM]

Fortran.3.31.0 Status and history
Original draft - DLN

Fortran.3.31.1 Language-specific terminology

A Fortran processor need not evaluate any part of an expression
not needed to compute the value of the expression. Side effects
of functions contributing to such portions of expressions
are processor-dependent, and any values associated with such
problematic evaluation is undefined.

Fortran.3.31.2 Description of application vulnerability

Fortran.3.31.3 Mechanism of failure

Fortran.3.31.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not put functions in expressions where they might not be
evaluated
if the function has side effects.
Prefer to use pure functions where that will achieve
the programming objective.

Fortran.3.31.5 Implications for standardization in Fortran

Fortran might consider an attribute to control function evaluation
in expressions.

Fortran.3.31.6 Bibliography

%%%%%

Fortran.3.32 Likely Incorrect Expression [KOA]

Fortran.3.32.0 Status and history
Original draft - DLN

Fortran.3.32.1 Language-specific terminology

Fortran.3.32.2 Description of application vulnerability

Fortran.3.32.3 Mechanism of failure

Fortran.3.32.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use variables to hold function results in complex expressions.

Fortran.3.32.5 Implications for standardization in Fortran

Fortran might consider an attribute to control function evaluation in expressions.

Fortran.3.32.6 Bibliography

%%%%%

Fortran.3.33 Dead and Deactivated Code [XYQ]

Fortran.3.33.0 Status and history
Original draft - DLN

Fortran.3.33.1 Language-specific terminology

Fortran.3.33.2 Description of application vulnerability

Fortran.3.33.3 Mechanism of failure

Fortran.3.33.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.33.5 Implications for standardization in Fortran

Fortran might consider requiring processors to detect and report the presence of unreachable code.

Fortran.3.33.6 Bibliography

%%%%%

Fortran.3.34 Switch Statements and Static Analysis [CLL]

Fortran.3.34.0 Status and history
Original draft - DLN

Fortran.3.34.1 Language-specific terminology

Fortran.3.34.2 Description of application vulnerability

Fortran.3.34.3 Mechanism of failure

Fortran.3.34.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.34.5 Implications for standardization in Fortran

Fortran might consider requiring processors to detect and report the presence of unreachable code.

Fortran.3.34.6 Bibliography

%%%%%

Fortran.3.35 Demarcation of Control Flow [EOJ]

Fortran.3.35.0 Status and history
Original draft - DLN

Fortran.3.35.1 Language-specific terminology

Fortran.3.35.2 Description of application vulnerability

Fortran.3.35.3 Mechanism of failure

Fortran.3.35.4 Avoiding the vulnerability or mitigating its effects in Fortran

Note that the non-block form of the DO construct is a decremental feature,
and as such, it should not be used.

Use the block form of the DO loop.

Fortran.3.35.5 Implications for standardization in Fortran

Fortran.3.35.6 Bibliography

%%%%%

Fortran.3.36 Loop Control Variables [TEX]

Fortran.3.36.0 Status and history
Original draft - DLN

Fortran.3.36.1 Language-specific terminology

It is not possible to modify the loop control variable
of a DO loop in any way that the processor can detect.
If the loop control variable is passed to a procedure,
the processor might not be able to detect violations.

Fortran.3.36.2 Description of application vulnerability

Fortran.3.36.3 Mechanism of failure

Fortran.3.36.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Fortran.3.36.5 Implications for standardization in Fortran

Fortran might consider an option to require all procedures called
with loop variables as arguments to have explicit interface and
argument intents, so the dummy argument receiving
the loop control variable can be checked to be intent(in).

Fortran.3.36.6 Bibliography

%%%%%

Fortran.3.37 Off-by-one Error [XZH]

Fortran.3.37.0 Status and history
Original draft - DLN

Fortran.3.37.1 Language-specific terminology

Fortran.3.37.2 Description of application vulnerability

Fortran.3.37.3 Mechanism of failure

Fortran.3.37.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Be clear about < versus <= and > versus >= operators.

Use inquiry intrinsics to determine the upper and lower bounds
of array extents.

Fortran.3.37.5 Implications for standardization in Fortran

Fortran.3.37.6 Bibliography

%%%%%

Fortran.3.38 Structured Programming [EWD]

Fortran.3.38.0 Status and history
Original draft - DLN

Fortran.3.38.1 Language-specific terminology

Fortran.3.38.2 Description of application vulnerability

Fortran.3.38.3 Mechanism of failure

Fortran.3.38.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not use alternate returns.
Do not use branches from input/output statements
when status (error or end) conditions occur.

Fortran.3.38.5 Implications for standardization in Fortran

Fortran might consider an option to forbid branching
from transfer statements, and alternate returns from subprograms.

Fortran.3.38.6 Bibliography

%%%%%

Fortran.3.39 Passing Parameters and Return Values [CSJ]

Fortran.3.39.0 Status and history
Original draft - DLN

Fortran.3.39.1 Language-specific terminology

Fortran does not specify a mechanism for passing values
into or out of a subroutine. This is governed by argument association
rules, together with argument intents.

Fortran.3.39.2 Description of application vulnerability

Fortran.3.39.3 Mechanism of failure

Fortran.3.39.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use argument intents.

Fortran.3.39.5 Implications for standardization in Fortran

Fortran might consider introducing a more complete set

of argument intents, to cover all cases including pointers to constants
versus constant pointers.

Fortran.3.39.6 Bibliography

%%%%%

Fortran.3.40 Dangling References to Stack Frames [DCM]

Fortran.3.40.0 Status and history
Original draft - DLN

Fortran.3.40.1 Language-specific terminology

Fortran.3.40.2 Description of application vulnerability

Fortran.3.40.3 Mechanism of failure

Fortran.3.40.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not assign local targets to pointers whose longevity
is longer than the function's execution.

Fortran.3.40.5 Implications for standardization in Fortran

Fortran might consider providing that processors shall detect
and report where pointers declared outside a procedure
are pointer assigned to procedure local targets.

Fortran.3.40.6 Bibliography

%%%%%

Fortran.3.41 Subprogram Signature Mismatch [OTR]

Fortran.3.41.0 Status and history
Original draft - DLN

Fortran.3.41.1 Language-specific terminology

Fortran.3.41.2 Description of application vulnerability

Fortran.3.41.3 Mechanism of failure

Fortran.3.41.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use modules to package procedures so they will have explicit interfaces.
Use interface bodies to describe external procedures, these might be

generated automatically by a tool, or by the processor.

Fortran.3.41.5 Implications for standardization in Fortran

Fortran might consider providing that processors shall have the ability
to require explicit interfaces for all procedures.

Fortran.3.41.6 Bibliography

%%%%%

Fortran.3.42 Recursion [GDL]

Fortran.3.42.0 Status and history
Original draft - DLN

Fortran.3.42.1 Language-specific terminology

Fortran.3.42.2 Description of application vulnerability

Fortran.3.42.3 Mechanism of failure

Fortran.3.42.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.42.5 Implications for standardization in Fortran

Fortran.3.42.6 Bibliography

%%%%%

Fortran.3.43 Returning Error Status [NZN]

Fortran.3.43.0 Status and history
Original draft - DLN

Fortran.3.43.1 Language-specific terminology

Fortran.3.43.2 Description of application vulnerability

Fortran.3.43.3 Mechanism of failure

Fortran.3.43.4 Avoiding the vulnerability or mitigating its effects in Fortran

Always check the STAT= or IOSTAT= specifier as appropriate.

Fortran.3.43.5 Implications for standardization in Fortran

Fortran might consider providing that processors shall have the ability

to require all statements supporting a STAT= or IOSTAT= specifier
to have one present on each occurrence.

Fortran.3.43.6 Bibliography

%%%%%

Fortran.3.44 Termination Strategy [REU]

Fortran.3.44.0 Status and history
Original draft - DLN

Fortran.3.44.1 Language-specific terminology

Fortran.3.44.2 Description of application vulnerability

Fortran.3.44.3 Mechanism of failure

Fortran.3.44.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Understand and use ALL STOP, STOP and SYNC IMAGES correctly.

Fortran.3.44.5 Implications for standardization in Fortran

Fortran.3.44.6 Bibliography

%%%%%

Fortran.3.45 Extra Intrinsics [LRM]

Fortran.3.45.0 Status and history
Original draft - DLN

Fortran.3.45.1 Language-specific terminology

Fortran.3.45.2 Description of application vulnerability

Fortran.3.45.3 Mechanism of failure

Fortran.3.45.4 Avoiding the vulnerability or mitigating its effects
in Fortran

Give all external names the external attribute,
or better, an explicit interface.

Fortran.3.45.5 Implications for standardization in Fortran

Fortran.3.45.6 Bibliography

%%%%%

Fortran.3.46 Type-breaking Reinterpretation of Data [AMV]

Fortran.3.46.0 Status and history
Original draft - DLN

Fortran.3.46.1 Language-specific terminology

Fortran.3.46.2 Description of application vulnerability

Fortran.3.46.3 Mechanism of failure

Fortran.3.46.4 Avoiding the vulnerability or mitigating its effects in Fortran

Do not rely on different names with different types in storage sequences.
Do not use TRANSFER.
Do not cause storage association of objects of different type by using common or equivalence.

Fortran.3.46.5 Implications for standardization in Fortran

Fortran might consider requiring processors to detect and report when type breaking reuse of bits occurs.

Fortran.3.46.6 Bibliography

%%%%%

Fortran.3.47 Memory Leak [XYL]

Fortran.3.47.0 Status and history
Original draft - DLN

Fortran.3.47.1 Language-specific terminology

Fortran.3.47.2 Description of application vulnerability

Fortran.3.47.3 Mechanism of failure

Fortran.3.47.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use allocatable or automatic variables in local procedures.
Do not apply the SAVE attribute.

Fortran.3.47.5 Implications for standardization in Fortran

Fortran.3.47.6 Bibliography

%%%%%

Fortran.3.48 Argument Passing to Library Functions [TRJ]

Fortran.3.48.0 Status and history
Original draft - DLN

Fortran.3.48.1 Language-specific terminology

Fortran.3.48.2 Description of application vulnerability

Fortran.3.48.3 Mechanism of failure

Fortran.3.48.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use explicit interfaces for libraries.
Use tools, if needed, to create interfaces for libraries.

Fortran.3.48.5 Implications for standardization in Fortran

Fortran.3.48.6 Bibliography

%%%%%

Fortran.3.49 Dynamically-linked Code and Self-modifying Code [NYY]

Fortran.3.49.0 Status and history
Original draft - DLN

Fortran.3.49.1 Language-specific terminology

Fortran does not support self-modifying code.

Fortran.3.49.2 Description of application vulnerability

Fortran.3.49.3 Mechanism of failure

Fortran.3.49.4 Avoiding the vulnerability or mitigating its effects in Fortran

Fortran.3.49.5 Implications for standardization in Fortran

Fortran.3.49.6 Bibliography

%%%%%

Fortran.3.50 Library Signature [NSQ]

Fortran.3.50.0 Status and history
Original draft - DLN

Fortran.3.50.1 Language-specific terminology

Fortran has no self-modifying code.

Fortran.3.50.2 Description of application vulnerability

Fortran.3.50.3 Mechanism of failure

Fortran.3.50.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use explicit interfaces for libraries.
Use tools, if needed, to create interfaces for libraries.

Fortran.3.50.5 Implications for standardization in Fortran

Fortran.3.50.6 Bibliography

%%%%%

Fortran.3.51 Unanticipated Exceptions from Library Routines [HJW]

Fortran.3.51.0 Status and history
Original draft - DLN

Fortran.3.51.1 Language-specific terminology

Fortran does not support handling exceptions from libraries.

Fortran.3.51.2 Description of application vulnerability

Fortran.3.51.3 Mechanism of failure

Fortran.3.51.4 Avoiding the vulnerability or mitigating its effects in Fortran

Use explicit interfaces for libraries.
Use tools, if needed, to create interfaces for libraries.

Fortran.3.51.5 Implications for standardization in Fortran

Fortran might consider providing a way to handle exceptions.

Fortran.3.51.6 Bibliography