ISO/IEC JTC 1/SC 22/OWGV N 0220

Draft Fortran Annex

Date	24 August 2009
Contributed by	Dan Nagle
Original file name	09-258r1.txt
Notes	Replaces N0216

```
To: WG23                              09-258r1
Subject: Draft Fortran Annex
From: Dan Nagle
Date: 2009 August 12

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


Annex Fortran

Vulnerability Descriptions for Fortran


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


Fortran.2 General Terminology
The Fortran standard specifies the forms Fortran programs
(source code) 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 requirement on the processor's ability to detect errors is limited
to those errors that can be detected by reference
to the numbered syntax rules
and constraints in the standard.
```

A behaviour not completely specified by ISO/IEC 1539-1
is said to be either processor dependent or undefined.

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

A Fortran processor optionally cooperates
with one or more processors, called companion processors,
that might be compilers of other languages.
A processor is its own companion processor,
additional companion processors might 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.
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
2009Jul14 - Updated in light of n0202 meeting 11 DLN
2009Jul15 - Updated in light of n0202 meeting 11 in plenary DLN
2009Aug12 - Updated during J3 189 DLN

Fortran.3.1.1 Terminology and features

<deleted feature>: A deleted feature is feature
that is not specified by the current revision of the standard,
but was specified by an earlier revision of the standard.

<obsolescent feature>: An obsolescent feature is feature
that is specified by the current revision of the standard;
its use is discouraged because a better alternative is specified
by the current revision of the standard.

<storage association>: Storage association is the association
of two or more data objects that occurs when two or more
storage sequences share or are aligned
with one or more storage units.

<storage sequence>: A storage sequence is a sequence
of storage units.

A storage unit is a numeric storage unit, a character storage unit,
a file storage unit, or an unspecified storage unit.

<common block>: A common block is a block of physical storage
that might be shared among scoping units.

<common statement>: A common statement specifies common block.

<save attribute>: The save attribute specifies that
a local data entity
retains its definition status between executions of the statements
of its scope.

<intrinsic procedures>: Intrinsic procedures
are procedures specified
by the standard or by a processor in an extended set.

<implicit type>: Where there is no explicit declaration for a name,
and implicit none is not specified,
the implicit type of a variable is the type
is implied by the first letter of its name.

Fortran.3.1.2 Description of vulnerability

The author of a program might have understood and used a feature
whose use was common at the time the program was written,
but the entirety of effects of some
features might not be known to modern programmers.
These effects might produce semantic results not in accord
with the modern programmer's expectations.
They might be beyond the knowledge of modern code reviewers.

For example, supplying an initial value in the declaration
of a local variable implies that the variable has the save attribute.

A common statement might specify more than one common block,
and the definition of a common block might be extended by a subsequent
common statement in the same program unit.

A common block might not retain its definition status
if it goes out of scope.

Names of different types may be storage associated.
Defining the name of one type usually causes the value
of the another type to become undefined. Storage associated sequences
of the same type might give aliases to storage locations.

If implicit typing is used, a simple spelling error might introduce
a new name, that is similar to the intended name.
The effect on program execution is unpredictable.

A programmer might not expect any of the above situations.

Fortran.3.1.3 Avoiding the vulnerability or mitigating its effects

- Use the processor to detect and identify extensions,
including processor-defined intrinsic procedures.
Avoid the use of processor extensions.

- Use the processor to detect and identify obsolescent or deleted features.
All such features have modern counterparts
that are safer, easier to understand, and more like to the semantics of other languages.

- Use a tool to detect any of the above.

- Use an interface body or a procedure declaration statement
to specify the external attribute for any external or dummy procedure.
Prefer to use module procedures in place of external procedures.
This will prevent the processor
from substituting its own intrinsic of the same signature.

- Place the entire specification of one common block entirely within
a single statement, or place the specification of one common block
in consecutive common statements.

- Do not specify more than one common block on a single statement.

- Use identical declarations of a common block in all scoping units
that access that common block.
Prefer the use of modules to share data among scoping units.

- Avoid the use of equivalence statements.

- Be aware that an initial value of a variable implies the save attribute.
Always specify the save attribute when supplying an initial value.

- Use IMPLICIT NONE to require explicit declarations.

Fortran.3.1.4 Implications for standardization

Future standardization efforts should consider:

- Identifying, deprecating, and replacing features
whose use is problematic and where there is a safer and clearer
alternative in the modern revisions of the language,
or in current practice in other languages.

Fortran.3.1.5 Bibliography

None.

%%%%%

Fortran.3.2 Unspecified Behaviour [BQF]

Fortran.3.2.0 Status and history

Original draft - DLN
2009Jul14 - Updated in light of n0202 meeting 11 DLN
2009Aug13 - Updated during J3 189 DLN

Fortran.3.2.1 Fortran-specific terminology and features

<Unspecified Behaviour>: The use of any form or relation not specified by the Fortran standard is unspecified. The use of any form or relationship given a meaning not specified by the standard is unspecified.
Any behaviour not specified by the standard is unspecified behaviour.

Fortran.3.2.2 Description of 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.

What a processor does with non-standard code is unpredictable.
The behaviour of non-standard code can change between processors.
It is entirely unpredictable.

Fortran.3.2.3 Avoiding the vulnerability or mitigating its effects

- Use processor options to detect and report use of non-standard features.
- Obtain diagnostics from more than one source, for example, use code checking tools.
- Avoid the use of non-standard intrinsic procedures.
- Specify the intrinsic attribute for all non-standard intrinsic procedures used.

Fortran.3.2.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Requiring that processors have the ability to detect and report the occurrence within a submitted program unit of extensions to standard intrinsic procedures.
- Requiring that processors have the ability to detect and report the occurrence within a submitted program unit of an invalid use of character constants as format specifiers.

Fortran.3.2.5 Bibliography

None.

%%%%%

Fortran.3.3 Undefined Behaviour [EWF]

Fortran.3.3.0 Status and history

Original draft - DLN
2009Jul14 - Updated in light of n0202 meeting 11 DLN
2009Aug13 - Updated during J3 189 DLN

This vulnerability is described
by Fortran.3.2 Unspecified Behaviour [BQF]
described above.

%%%%%

Fortran.3.4 Implementation-defined Behaviour [FAB]

Fortran.3.4.0 Status and history

Original draft - DLN
2009Jul14 - Updated in light of n0202 meeting 11 DLN
2009Aug12 - Updated during J3 189 DLN

Fortran.3.4.1 Fortran-specific terminology and features

<Processor-dependent>: Processor-dependent behaviour is the Fortran
term for implementation-defined behaviour.
See Annex A.2 of ISO/IEC 1539-1 (2010) for a list
of processor dependencies.

Fortran.3.4.2 Description of vulnerability

Different processors might process processor-dependencies
differently. Relying on one behaviour is not guaranteed
by the Fortran standard.

Reliance on one behaviour where the standard explicitly allows
several is not portable. The behaviour is liable to change between
different processors.

Fortran.3.4.3 Avoiding the vulnerability or mitigating its effects

- Do not rely on processor dependencies. See Annex A.2

for a complete list.

Fortran.3.4.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Requiring that processors have the ability to detect and report the occurrence within a submitted program unit of more situations where the appearance of depending upon processor-dependent behaviour occurs. For example, consider requiring that processors have the ability to detect and report the occurrence within a submitted program unit of the appearance of a literal kind type parameter value.

Fortran.3.4.5 Bibliography

None.

%%%%%

Fortran.3.5 Deprecated Language Features [MEM]

Fortran.3.5.0 Status and history

Original draft - DLN
2009Jul14 - Updated in light of n0202 meeting 11 DLN
2009Aug12 - Updated during J3 189 DLN

Fortran.3.5.1 Fortran-specific terminology and features

<deleted feature>: A deleted feature is a feature that is not specified by the current revision of the standard, but was specified by an earlier revision of the standard.

<obsolescent feature>: An obsolescent feature is a feature that is specified by the current revision of the standard; its use is discouraged because a better alternative is specified by the current revision of the standard.

Fortran.3.5.2 Description of vulnerability

The author of a program might have understood and used a feature whose use was common at the time the program was written, but the entirety of effects of some features might not be known to modern programmers. These effects might produce semantic results not in accord with the modern programmer's expectations. They might be beyond the knowledge of modern code reviewers.

See Annex B of ISO/IEC 1539-1 (2010) for a complete list.

Fortran.3.5.3 Avoiding the vulnerability or mitigating its effects

- Do not use obsolescent or deleted features.

- Use the processor to detect and identify obsolescent or deleted features;
then replace them with a modern synonym.

- Use processor options to require adherence to the latest standard.

- Use a tool to accomplish any of the above.

Fortran.3.5.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Identifying, deprecating, and replacing features
whose use is problematic and where there is a safer and clearer
alternative in the modern revisions of the language,
or in current practice in other languages.

Fortran.3.5.5 Bibliography

None.

%%%%%

Fortran.3.6 Pre-processor Directives [NMP]

Fortran.3.6.0 Status and history
Original draft - DLN
2009Jul14 - Updated in light of n0202 meeting 11 DLN

ISO/IEC 1539-1 (2010) does not specify a preprocessor.

%%%%%

Fortran.3.7 Choice of Clear Names [NAI]

Fortran.3.7.0 Status and history
Original draft - DLN
2009Jul14 - Updated in light of n0202 meeting 11 DLN
2009Aug12 - Updated during J3 189 DLN

Fortran.3.7.1 Fortran-specific terminology and features

<processor character set>: The processor character set is divided
into control characters and graphic characters.
The graphic characters
are further divided into letters, digits, the underscore,
special characters (Table 3.1), and other characters.

<implicit type>: Where there is no explicit declaration for a name,
the implicit type of a variable is the type
given by the first letter of its name.

<character context>: Character context is a character literal constant
or a character string edit descriptor.

Fortran.3.7.2 Description of vulnerability

The set of other characters supported by a processor
is processor-dependent.

Fortran is a single case language; upper case and lower case
are treated identically by the standard except
in a character context.

Names differing only in capitalization are not distinct.
While some processors have options
to preserve case or distinguish names by case, others do not.

While the underscore might be used in names, the consecutive
underscores is difficult to readily see.

Fortran has keywords but no reserved words.
Some keywords allow an optional blank,
as described in subclause 3.3.2.2.

When implicit typing is in effect, a misspelling of a name
results in a new variable. Use of implicit none prohibits
implicit typing.

Fortran.3.7.3 Avoiding the vulnerability or mitigating its effects

- Always use free form so blanks are significant.

- Always use the optional blank within a keyword (3.3.2.2).

- Do not try to distinguish names by case only.

- Do not use consecutive underscores in a name.

- Do not use keywords as names.

- Always use implicit none to disable implicit typing.

Fortran.3.7.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Requiring that processors have the ability to detect and report the occurrence within a submitted program unit of names appearing to differ only in case, or that have consecutive underscores.

- Requiring that processors have the ability to detect and report the occurrence within a submitted program unit of names indistinguishable from keywords.

- Requiring that processors have the ability to detect and report the occurrence within a submitted program unit of keywords that could have embedded blanks, but do not.

Fortran.3.7.5 Bibliography

None.

%%%%%

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

Fortran.3.8.0 Status and history

Original draft - DLN
2009Jul16 - Upgrade to new format DLN
2009Aug13 - Upgrade during J3 189 DLN

Fortran.3.8.1 Language-specific terminology

<character assignment>: Character assignment to a fixed length variable has a truncate or blank fill semantic. This does not apply to allocatable character variables, where the length is adjusted as needed.

Fortran.3.8.2 Description of vulnerability

File names appearing as input to OPEN and INQUIRE statements, and character values appearing as input in references to the GET_ENVIRONMENT_VARIABLE intrinsic and the EXECUTE_COMMAND_LINE intrinsic, without regard to trailing blanks.

This is a result of the historic use of static length characters, where a character assignment might leave trailing blanks. Thus, trailing blanks are not used externally.

Any character literal constant specified on an INCLUDE line, or any file name, or any environment variable name is processor-dependent.

Fortran.3.8.3 Avoiding the vulnerability or mitigating its effects

- Do not attempt to distinguish file names or other external identifiers by trailing blanks.

- Prefer not to use control characters, special characters, or other characters to distinguish file names or other external identifiers.

Fortran.3.8.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing a mechanism whereby trailing blanks might be used externally.

- Providing a means to determine the characteristics distinguishing file names and other external identifiers.

Fortran.3.8.5 Bibliography

None.

%%%%%

Fortran.3.9 Unused Variable [XYR]

Fortran.3.9.0 Status and history
Original draft - DLN
2009Jul17 - Upgrade to new format DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.9.1 Language-specific terminology

<implicit type>: Where there is no explicit declaration for a name, and implicit none is not specified, the implicit type of a variable is the type is implied by the first letter of its name.

<use association>: Use association is association between entities in a module and entities in a scoping unit that references that module.

Fortran.3.9.2 Description of vulnerability

If implicit typing is used, a simple spelling error might introduce a new name, that is similar to the intended name. The effect on program execution is unpredictable.

A common block declaration might contain unused names.

A use statement, without an only clause,
might expose unused names.

Fortran.3.9.3 Avoiding the vulnerability or mitigating its effects

- Use IMPLICIT NONE to require explicit declarations.

- Use processor options to report unused variables.

- Do not use common blocks, as the common might legitimately contain names of variables unused in one scoping unit.

- Use ONLY clauses on USE statements to ensure that only those names intended are accessed.

Fortran.3.9.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing that processors have the ability to detect and report the accessibility within a submitted program unit of an unused name.

- Providing that processors have the ability to detect and report the occurrence within a submitted program unit of any name given an implicit type.

Fortran.3.9.5 Bibliography

None.

%%%%%

Fortran.3.10 Identifier Name Reuse [YOW]

Fortran.3.10.0 Status and history

Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.10.1 Language-specific terminology

<processor character set>: The processor character set is divided into control characters and graphic characters.
The graphic characters
are further divided into letters, digits, the underscore,
special characters (Table 3.1), and other characters.

<implicit type>: Where there is no explicit declaration for a name,

and implicit none is not specified,
the implicit type of a variable is the type
is implied by the first letter of its name.

<host association>: Association between a name in a contained scope
and the host scope, or between a submodule and the module.

<use association>: Use association is association between entities
in a module and entities in a scoping unit that references that module.

Fortran.3.10.2 Description of vulnerability

All characters in a Fortran name are significant.

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

In addition to the vulnerabilities in 6.10, additional similar
vulnerabilities arise from the accessibility of names from modules.

Fortran.3.10.3 Avoiding the vulnerability or mitigating its effects

- Avoid the use of BLOCKS.

- Avoid the use of type specifications in forall headers.

- Use an ONLY clause on each USE statement.

- Check that each name declared in a contained scope
does not appear in the host scope.

- Prefer argument association instead of host association
in internal procedures.

Fortran.3.10.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing that processors have the ability to detect and report
the occurrence within a submitted program unit of any instance
where a declaration in a nested scope prevents access to a name
in the host scope.

- Providing that processors have the ability to detect and report
the occurrence within a submitted program unit of any instance
where a name is accessed by host association.

- Adding a means to control which, if any, names in a contained scope
are accessible from its host.

- Decrementing BLOCKS and declarations in forall headers.

Fortran.3.10.5 Bibliography

None.

%%%%%

Fortran.3.11 Type System [IHN]

Fortran.3.11.0 Status and history

Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.11.1 Language-specific terminology

<intrinsic type>: An intrinsic type is a type defined by the standard to be an intrinsic type.

<derived type>: A derived type is a type defined by the program, or a type defined by the standard to be a defined modules.

Fortran.3.11.2 Description of vulnerability

Fortran promotes operands in expressions from smaller precision or range to larger precision or range within a numeric type, and among types, from integer to real to complex. If an integer expression
contains operands of real or complex type, its value might depend upon the optimization level or rounding mode in effect.

Fortran expressions are evaluated without regard to context; in an assignment, and in a few other situations, the type of an expression is converted as needed.

There is no automatic conversion between derived types and intrinsic types.
There is no automatic conversion between derived types.

The is no automatic conversion of an actual argument to the type of the dummy argument.
Where procedures do not have explicit interfaces, the type of an actual argument might not correspond to the dummy argument, leading to an incorrect interpretation of the argument.

Fortran.3.11.3 Avoiding the vulnerability or mitigating its effects

- Explicitly convert expressions as needed.

- Use explicit interfaces.

- Use a tool to detect any of the above.

Fortran.3.11.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing a capability to report, forbid,
or control automatic conversions.

- Providing an inquiry intrinsic to provide the largest integer
a real type is capable of representing exactly.

- Providing the ability to detect and report integer overflow.

Fortran.3.11.5 Bibliography

None.

%%%%%

Fortran.3.12 Bit Representations [STR]

Fortran.3.12.0 Status and history

Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.12.1 Language-specific terminology

<storage association>: Storage association is the association
of two or more data objects that occurs when two or more
storage sequences share or are aligned
with one or more storage units.

<storage sequence>: A storage sequence is a sequence
of storage units.
A storage unit is a numeric storage unit, a character storage unit,
a file storage unit, or an unspecified storage unit.

Fortran.3.12.2 Description of vulnerability

A bit representation might be made visible when a location
in a storage sequence has names with different types.
This can occur
when a storage location in a common block has different types

in different scopes, or when an equivalence set has names
of different types.
The TRANSFER intrinsic copies bits between variables
without regard for their types.

Using binary representations in integer and real conversion intrinsics
might not be portable.

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

Fortran.3.12.3 Avoiding the vulnerability or mitigating its effects

- 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 converting pointers between
Fortran and C representations.

- Use a tool to detect any of the above.

Fortran.3.12.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing that processors have the ability to detect and report
the occurrence within a submitted program unit
of storage locations with more then one type, and to report the use
of the TRANSFER intrinsic.

Fortran.3.12.5 Bibliography

None.

%%%%%

Fortran.3.13 Floating-point Arithmetic [PLF]

Fortran.3.13.0 Status and history
Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.13.1 Language-specific terminology

<inquiry intrinsic>: An inquiry intrinsic is an intrinsic function
that returns characteristics of the numeric model of the type
of its argument.

<real>: The real type has values that approximate the mathematical real
numbers. There are at least two kinds of type real.

<complex>: The complex type has values that approximate
the mathematical complex numbers. There is a complex kind
for each real kind.

Fortran.3.13.2 Description of application vulnerability

Possibly distinct kinds of type real might be available
from the IEEE intrinsic modules.

Fortran.3.13.3 Avoiding the vulnerability or mitigating its effects

- Use the IEEE intrinsic modules, and the values
and procedures therein.

Fortran.3.13.4 Implications for standardization in Fortran

None beyond C6.13.6.

Fortran.3.13.5 Bibliography

None.

%%%%%

Fortran.3.14 Enumerator Issues [CCB]

Fortran.3.14.0 Status and history
Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.14.1 Language-specific terminology

<enumerator>: An enumerator is a named constant of integer type.
The kind of the integer type depends upon the companion processor.

Fortran.3.14.2 Description of vulnerability

The Fortran enumerator type is an integer type; it is not a new type.
The Fortran enumerator type is designed primarily for interoperability
with the C enumerator type.

Fortran.3.14.3 Avoiding the vulnerability or mitigating its effects

- Do not use enumerators for purposes
other than interoperability with C.

Fortran.3.14.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Defining a standard enumerator type
for uses beyond interoperability with C.

Fortran.3.14.5 Bibliography

None.

%%%%%

Fortran.3.15 Numeric Conversion Errors [FLC]

Fortran.3.15.0 Status and history

Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.15.1 Language-specific terminology

None.

Fortran.3.15.2 Description of vulnerability

As per Clause 6.

Fortran.3.15.3 Avoiding the vulnerability or mitigating its effects

As per Clause 6.

Fortran.3.15.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding a requirement that processors have

the ability to detect and report conversions that might result
in loss of data.

- Adding an inquiry intrinsic to report the largest integer that
a real type can represent exactly.

Fortran.3.15.5 Bibliography

None.

%%%%%

Fortran.3.16 String Termination [CJM]

Fortran.3.16.0 Status and history

Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran does not use a string termination code.

%%%%%

Fortran.3.17 Boundary Beginning Violation [XYX]

Fortran.3.17.0 Status and history

Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

This is covered by [XYZ]

%%%%%

Fortran.3.18 Unchecked Array Indexing [XYZ]

Fortran.3.18.0 Status and history

Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.18.1 Language-specific terminology

<extent>: An extent of an array is the number of elements
in a single dimension of an array.

<rank>: Rank is the number of array dimensions of a data entity

(zero for a scalar entity).

<upper bound>: The upper bound is the largest valid index
of a dimension.

<lower bound>: The lower bound is the smallest valid index
of a dimension.

<array assignment>: Array assignment allows one array
to be assigned to another.

Fortran.3.18.2 Description of vulnerability

An array access using an index outside the bounds of any dimension
is prohibited, its effects are unpredictable.

The upper bound of the last dimension of an assumed size dummy argument
might not be known, and this might defeat bounds checking.

Fortran.3.18.3 Avoiding the vulnerability or mitigating its effects

- Use whole array assignment, 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.

Fortran.3.18.4 Implications for standardization in Fortran

None.

Fortran.3.18.5 Bibliography

None.

%%%%%

Fortran.3.19 Unchecked Array Copying [XYW]

Fortran.3.19.0 Status and history
Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

This is covered by [XYZ].

%%%%%

Fortran.3.20 Buffer Overflow [XZB]

Fortran.3.20.0 Status and history
Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

This is covered by [XYZ].

%%%%%

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

Fortran.3.21.0 Status and history
Original draft - DLN
2009Jul17 Upgrade to new format - DLN

A Fortran pointer cannot change type.

%%%%%

Fortran.3.22 Pointer Arithmetic [RVG]

Fortran.3.22.0 Status and history
Original draft - DLN
2009Jul17 Upgrade to new format - DLN

There is no arithmetic on pointers.

%%%%%

Fortran.3.23 Null Pointer Dereference [XYH]

Fortran.3.23.0 Status and history
Original draft - DLN
2009Jul17 Upgrade to new format - DLN
2009Aug13 - Upgraded during J3 189 DLN

Fortran.3.23.1 Language-specific terminology

<target>: A target is a data object that can be
associated with a pointer. The target is the actual storage.

<pointer>: Pointer is a data pointer or a procedure pointer.
A pointer might be associated, disassociated, or have undefined
association status.

<data pointer>: A data pointer is a data object that can be associated
with different targets during execution.

<procedure pointer>: A procedure pointer is a entity that can be
associated with different procedures during execution.

<pointer assignment>: Pointer assignment changes the association
status of a pointer.

<assignment>: For a pointer associated with a target, assignment
to the pointer affects the value of the target. For a pointer
associated with a target, a reference to the pointer provides
the value of the target.

<allocatable>: Allocatable is an attribute of a object that
can be allocated with varying properties during execution.
An allocatable object is either allocated or deallocated.

<associated>: When a pointer is associated with a target,
the pointer can be used to change the value of the target.

<allocated>: When an allocatable object is allocated,
it has its own storage.

Fortran.3.23.2 Description of application vulnerability

If a pointer that is not associated is referenced or defined,
the effects are undefined and unpredictable.

If an unallocated allocatable is referenced or defined,
the effects are undefined and unpredictable.

Fortran.3.23.3 Avoiding the vulnerability or mitigating its effects

- Use the associated intrinsic to ensure that a pointer
is associated with a target before referencing the name
with the pointer attribute.

- Use the allocated intrinsic to ensure that an allocatable
is allocated before referencing the name.

- Use allocatables in preference to pointers wherever allocatables
provide the desired functionality.

Fortran.3.23.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding new dummy argument intents to cover all cases
of pointer can change, or the target can change.

Fortran.3.23.5 Bibliography

None.

%%%%%

Fortran.3.24 Dangling Reference to Heap [XYK]

Fortran.3.24.0 Status and history
Original draft - DLN
2009Jul18 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.24.1 Language-specific terminology

<target>: A target is a data object that can be
associated with a pointer. The target is the actual storage.

<pointer>: Pointer is a data pointer or a procedure pointer.
A pointer might be associated, disassociated, or have undefined
association status.

<data pointer>: A data pointer is a data object that can be associated
with different targets during execution.

<procedure pointer>: A procedure pointer is a entity that can be
associated with different procedures during execution.

<pointer assignment>: Pointer assignment changes the association
status of a pointer.

<allocatable>: Allocatable is an attribute of a object that
can be allocated with varying properties during execution.
An allocatable object is either allocated or deallocated.

<save attribute>: The save attribute specifies that
a local data entity
retains its definition status between executions of the statements
of its scope.

Fortran.3.24.2 Description of vulnerability

Fortran does not specify how variables are stored.
Many implementations will store allocatable variables
or allocated pointer targets on the heap.

Fortran.3.24.3 Avoiding the vulnerability or mitigating its effects

- Use allocatables in preference to pointers wherever allocatables provide the desired functionality.

- Avoid pointer associating pointers declared outside the scope,
or that has the save attribute,
with an unsaved allocatable variables declared inside the scope.

- Apply the SAVE attribute to allocatable local variables
whenever a pointer declared outside the scope,
or that has the save attribute,
is pointer assigned to the variable.

Fortran.3.24.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding a requirement that processors have
the ability to detect and report
unit of where a pointer declared the scope is pointer assigned
target within a scope.

Fortran.3.24.5 Bibliography

None.

%%%%%

Fortran.3.25 Templates and Generics [SYM]

Fortran.3.25.0 Status and history
Original draft - DLN
2009Jul18 Upgrade to new format - DLN

This subclause does not apply to Fortran.

%%%%%

Fortran.3.26 Inheritance [RIP]

Fortran.3.26.0 Status and history
Original draft - DLN
2009Jul18 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.26.1 Language-specific terminology

<extends>: A child type derived from a parent type extends
the parent type.

<non_overridable attribute>: The non_overridable attribute specifies
that a binding shall not override an inherited binding.

<private attribute>: The private attribute specifies that a component
name is accessible only within the module containing the definition,
and within its descendants.

Fortran.3.26.2 Description of vulnerability

Fortran supports single inheritance only;
none in addition to those listed in subclause 6.26.

Fortran.3.26.3 Avoiding the vulnerability or mitigating its effects

- Use the non_overridable attribute whenever possible.

- Apply the private attribute to components of the parent type.

Fortran.3.26.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding class invariants.

Fortran.3.26.5 Bibliography

None.

%%%%%

Fortran.3.27 Initialization of Variables [LAV]

Fortran.3.27.0 Status and history
Original draft - DLN
2009Jul19 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.27.1 Language-specific terminology

<save attribute>: The save attribute specifies that
a local data entity
retains its definition status between executions of the statements
of its scope.

Fortran.3.27.2 Description of vulnerability

Specifying an initial value for a variable implies the save attribute.

Fortran.3.27.3 Avoiding the vulnerability or mitigating its effects

- Use executable statements to supply initial values to variables
when the save attribute is not required.

- Use named constants if the value will not change.

Fortran.3.27.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing a way to declare an initial value
for a variable every time the scope in which it is declared
is entered without forcing static storage.

- Adding a requirement that processors have
the ability to detect and report a variable having the save attribute
without an explicit declaration of the attribute.

Fortran.3.27.5 Bibliography

None.

%%%%%

Fortran.3.28 Wrap-around Error [XYY]

Fortran.3.28.0 Status and history
Original draft - DLN
2009Jul19 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.28.1 Language-specific terminology

<numeric model>: The processor-supplied model
of the numeric representation supported by the processor.
Its parameters are available via the inquiry intrinsics.

Fortran.3.28.2 Description of application vulnerability

Fortran provides no description of wrap-around error of integer overflow,
nor any way to detect it.

Fortran.3.28.3 Avoiding the vulnerability or mitigating its effects

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

- Prefer the use of the largest integer type available where
results might overflow.

- Use inquiry intrinsics to determine the range supported
by the type used.

Fortran.3.28.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing a separate type for bit operations.

- Providing an means to predict and/or detect when
an integer operation will produce or has produced a wrap-around error.

Fortran.3.28.5 Bibliography

None.

%%%%%

Fortran.3.29 Sign Extension Error [XZI]

Fortran.3.29.0 Status and history
Original draft - DLN
2009Jul19 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.29.1 Language-specific terminology

<BOZ literal constant>: A BOZ literal constant is a binary, octal
or hexadecimal literal constant.

Fortran does not have unsigned data types.

Fortran.3.29.2 Description of vulnerability

As per subclause 6.29.

Fortran.3.29.3 Avoiding the vulnerability or mitigating its effects

- Choose the particular bit shift intrinsic as appropriate.

- Understand the bit model, and the effects of BOZ literal constants.

- Avoid using the transfer intrinsic.

- Prefer the use of the bit-level intrinsics to BOZ literal constants.

Fortran.3.29.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing a separate type for bit operations.

Fortran.3.29.5 Bibliography

None.

%%%%%

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

Fortran.3.30.0 Status and history
Original draft - DLN
2009Jul19 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.30.1 Language-specific terminology

None.

Fortran.3.30.2 Description of vulnerability

Assignment is not an operator in Fortran.

Bit operations are intrinsic functions, so precedence is clear.

The precedence of unary negation is lower than the precedence
of exponentiation. This may be contrary to programmer's expectations.

Fortran.3.30.3 Avoiding the vulnerability or mitigating its effects

- Use parenthesis for clarity.

Fortran.3.30.4 Implications for standardization in Fortran

None.

Fortran.3.30.5 Bibliography

None.

%%%%%

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

Fortran.3.31.0 Status and history

Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.31.1 Language-specific terminology

<expression>: An expression represents either a data reference
or a computation, and its value is either a scalar or an array.
An expression is formed from operands, operators, and parentheses.

Fortran.3.31.2 Description of vulnerability

A Fortran processor might evaluate an expression in any order it chooses.

A Fortran processor need not evaluate any part of an expression
if it is 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 are undefined.

Fortran.3.31.3 Avoiding the vulnerability or mitigating its effects

- Prefer to use a subroutine where side effects
are needed for the programming objective.

- Prefer to use pure functions where that will achieve
the programming objective.

- Use parenthesis and/or partial result variables to control
the order of evaluation of expressions.

Fortran.3.31.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding a means to control function evaluation in expressions.

- Adding a means to control the order of evaluation
of procedure arguments.

Fortran.3.31.5 Bibliography

None.

%%%%%

Fortran.3.32 Likely Incorrect Expression [KOA]

Fortran.3.32.0 Status and history
Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

This is covered by [SAM] above.

%%%%%

Fortran.3.33 Dead and Deactivated Code [XYQ]

Fortran.3.33.0 Status and history
Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.33.1 Language-specific terminology

None.

Fortran.3.33.2 Description of vulnerability

As per subclause 6.33.

Fortran.3.33.3 Avoiding the vulnerability or mitigating its effects

As per subclause 6.33.

Fortran.3.33.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Requiring processors to detect and report
the occurrence within a submitted program unit of unreachable code.

Fortran.3.33.5 Bibliography

None.

%%%%%

Fortran.3.34 Switch Statements and Static Analysis [CLL]

Fortran.3.34.0 Status and history
Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.34.1 Language-specific terminology

None.

Fortran.3.34.2 Description of vulnerability

As per subclause 6.34.

Fortran.3.34.3 Avoiding the vulnerability or mitigating its effects

As per subclause 6.34.

Fortran.3.34.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Requiring processors to have the ability to detect and report
the occurrence within a submitted program unit where
a select case construct does not cover all cases.

- Requiring processors to have the ability to detect and report
the occurrence within a submitted program unit where
a select case construct contains a default block.

- Requiring processors to have the ability to detect and report
the occurrence within a submitted program unit where
a select type construct does not cover all cases.

- Requiring processors to have the ability to detect and report
the occurrence within a submitted program unit where
a select type construct contains a default block.

Fortran.3.34.5 Bibliography

None.

%%%%%

Fortran.3.35 Demarcation of Control Flow [EOJ]

Fortran.3.35.0 Status and history
Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.35.1 Language-specific terminology

None.

Fortran.3.35.2 Description of vulnerability

A shared terminal statement of a non-block do construct
might be difficult to understand.

Fortran.3.35.3 Avoiding the vulnerability or mitigating its effects

- Note that the non-block form of the do construct is
an obsolescent feature, and as such, it should not be used.

- Use the block form of the do loop.

Fortran.3.35.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Deleting the non-block form of the do construct.

Fortran.3.35.5 Bibliography

None.

%%%%%

Fortran.3.36 Loop Control Variables [TEX]

Fortran.3.36.0 Status and history
Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.36.1 Language-specific terminology

None.

Fortran.3.36.2 Description of vulnerability

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.

A loop iteration count calculation might silently overflow.

Fortran.3.36.3 Avoiding the vulnerability or mitigating its effects

- Use explicit interfaces for all routines called from within loops;
pass the loop variable only to intent(in) arguments.

Fortran.3.36.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Requiring an option that 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.3.36.5 Bibliography

None.

%%%%%

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

Fortran.3.37.0 Status and history
Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.37.1 Language-specific terminology

None.

Fortran.3.37.2 Description of vulnerability

As per subclause 6.37.

Fortran.3.37.4 Avoiding the vulnerability or mitigating its effects

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

Fortran.3.37.4 Implications for standardization in Fortran

None.

Fortran.3.37.5 Bibliography

None.

%%%%%

Fortran.3.38 Structured Programming [EWD]

Fortran.3.38.0 Status and history

Original draft - DLN
2009Jul20 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.38.1 Language-specific terminology

<alternate return>: Alternate return is a means
for a subroutine to return
to a point in the caller other than immediately
following the point of reference.

Fortran.3.38.2 Description of vulnerability

Alternate returns, and branches on error or end conditions
on input/output statements, are unstructured.

Fortran.3.38.3 Avoiding the vulnerability or mitigating its effects

- Do not use alternate returns.

- Do not use end, eor, or err specifiers in input/output statements.

- Use an iostat and/or iomsg specifier to check status.

Fortran.3.38.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding an option to forbid branching
from input/output statements, and alternate returns
from subprograms.

Fortran.3.38.5 Bibliography

None.

%%%%%

Fortran.3.39 Passing Parameters and Return Values [CSJ]

Fortran.3.39.0 Status and history

Original draft - DLN
2009Jul21 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.39.1 Language-specific terminology

<dummy argument>: The dummy argument is the name
of a procedure argument local to the procedure.

<actual argument>: The actual argument is the data entity passed
to the procedure when it is referenced.

<argument intent>: An argument intent is an attribute specifying
what actions might be made of a dummy argument.

Fortran.3.39.2 Description of application vulnerability

Fortran does not specify a particular argument passing mechanism.
This is governed by argument association rules,
together with argument intents,
and the contiguous, optional and value attributes.

Fortran.3.39.3 Avoiding the vulnerability or mitigating its effects

- Do not depend upon a particular argument passing mechanism.

- Specify argument intents.

- Specify argument attributes as needed.

Fortran.3.39.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding a more complete set of argument intents,
to cover all cases including distinguishing
actions on the pointer association status
and the pointer's target.

Fortran.3.39.5 Bibliography

None.

%%%%%

Fortran.3.40 Dangling References to Stack Frames [DCM]

Fortran.3.40.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN
2009Aug14 - Upgraded during J3 189 DLN

Fortran.3.40.1 Language-specific terminology

<target>: A target is a data object that can be
associated with a pointer. The target is the actual storage.

<pointer>: Pointer is a data pointer or a procedure pointer.
A pointer might be associated, disassociated, or have undefined
association status.

<data pointer>: A data pointer is a data object that can be associated
with different targets during execution.

<procedure pointer>: A procedure pointer is a entity that can be
associated with different procedures during execution.

<pointer assignment>: Pointer assignment changes the association
status of a pointer.

<allocatable>: Allocatable is an attribute of a object that
can be allocated with varying properties during execution.
An allocatable object is either allocated or deallocated.

<save attribute>: The save attribute specifies that
a local data entity
retains its definition status between executions of the statements
of its scope.

Fortran.3.40.2 Description of vulnerability

Fortran does not specify how variables are stored.
Many implementations will store local variables on the stack.

Fortran.3.40.3 Avoiding the vulnerability or mitigating its effects

- Use allocatables in preference to pointers wherever allocatables provide the desired functionality.

- Avoid pointer associating a pointer declared outside the scope,
or that has the save attribute,
with an unsaved local variable declared inside the scope.

- Apply the SAVE attribute to local variables
whenever a pointer declared outside the scope,
or that has the save attribute,
is pointer assigned to the variable.

Fortran.3.40.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Adding a requirement that processors have
the ability to detect and report
unit of where a pointer declared the scope is pointer assigned
target within a scope.

Fortran.3.40.5 Bibliography

None.

%%%%

Fortran.3.41 Subprogram Signature Mismatch [OTR]

Fortran.3.41.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.41.1 Language-specific terminology

<interface>:

Fortran.3.41.2 Description of vulnerability

Fortran.3.41.3 Avoiding the vulnerability or mitigating its effects

- 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.3.41.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing that processors shall have the ability
to require explicit interfaces for all procedures.

Fortran.3.41.5 Bibliography

None.

%%%%%

Fortran.3.42 Recursion [GDL]

Fortran.3.42.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.42.1 Language-specific terminology

Fortran.3.42.2 Description of vulnerability

Fortran.3.42.3 Avoiding the vulnerability or mitigating its effects

Fortran.3.42.4 Implications for standardization in Fortran

Future standardization efforts should consider:

Fortran.3.42.5 Bibliography

None.

%%%%%

Fortran.3.43 Returning Error Status [NZN]

Fortran.3.43.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.43.1 Language-specific terminology

Fortran.3.43.2 Description of vulnerability

Fortran.3.43.3 Avoiding the vulnerability or mitigating its effects

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

Fortran.3.43.4 Implications for standardization in Fortran

Future standardization efforts should 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.

- Providing that no statement would have a end= or err= specifier.

Fortran.3.43.5 Bibliography

None.

%%%%%

Fortran.3.44 Termination Strategy [REU]

Fortran.3.44.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.44.1 Language-specific terminology

Fortran.3.44.2 Description of vulnerability

Fortran.3.44.3 Avoiding the vulnerability or mitigating its effects

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

Fortran.3.44.4 Implications for standardization in Fortran

Future standardization efforts should consider:

Fortran.3.44.5 Bibliography

None.

%%%%%

Fortran.3.45 Extra Intrinsics [LRM]

Fortran.3.45.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.45.1 Language-specific terminology

Fortran.3.45.2 Description of vulnerability

Fortran.3.45.3 Avoiding the vulnerability or mitigating its effects

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

Fortran.3.45.4 Implications for standardization in Fortran

Future standardization efforts should consider:

Fortran.3.45.5 Bibliography

None.

%%%%%

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

Fortran.3.46.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.46.1 Language-specific terminology

Fortran.3.46.2 Description of vulnerability

Fortran.3.46.3 Avoiding the vulnerability or mitigating its effects

- 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.3.46.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Requiring processors to detect and report
when type breaking reuse of bits occurs.

Fortran.3.46.5 Bibliography

None.

%%%%%

Fortran.3.47 Memory Leak [XYL]

Fortran.3.47.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.47.1 Language-specific terminology

Fortran.3.47.2 Description of vulnerability

Fortran.3.47.3 Avoiding the vulnerability or mitigating its effects

- Use allocatable or automatic variables in local procedures.

- Do not apply the SAVE attribute.

Fortran.3.47.4 Implications for standardization in Fortran

Future standardization efforts should consider:

Fortran.3.47.5 Bibliography

None.

%%%%%

Fortran.3.48 Argument Passing to Library Functions [TRJ]

Fortran.3.48.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.48.1 Language-specific terminology

Fortran.3.48.2 Description of vulnerability

Fortran.3.48.3 Avoiding the vulnerability or mitigating its effects

- Use explicit interfaces for libraries.

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

Fortran.3.48.4 Implications for standardization in Fortran

Future standardization efforts should consider:

Fortran.3.48.5 Bibliography

None.

%%%%

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

Fortran.3.49.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.49.1 Language-specific terminology

Fortran.3.49.2 Description of vulnerability

Fortran.3.49.3 Avoiding the vulnerability or mitigating its effects

Fortran.3.49.4 Implications for standardization in Fortran

Future standardization efforts should consider:

Fortran.3.49.5 Bibliography

None.

%%%%

Fortran.3.50 Library Signature [NSQ]

Fortran.3.50.0 Status and history

Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.50.1 Language-specific terminology

Fortran.3.50.2 Description of vulnerability

Fortran.3.50.3 Avoiding the vulnerability or mitigating its effects

- Use explicit interfaces for libraries.

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

Fortran.3.50.4 Implications for standardization in Fortran

Future standardization efforts should consider:

Fortran.3.50.5 Bibliography

None.

%%%%%

Fortran.3.51 Unanticipated Exceptions from Library Routines [HJW]

Fortran.3.51.0 Status and history
Original draft - DLN
2009Jul21 Upgrade to new format - DLN

Fortran.3.51.1 Language-specific terminology

Fortran.3.51.2 Description of vulnerability

Fortran.3.51.3 Avoiding the vulnerability or mitigating its effects

- Use explicit interfaces for libraries.

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

Fortran.3.51.4 Implications for standardization in Fortran

Future standardization efforts should consider:

- Providing a way to handle exceptions.

Fortran.3.51.5 Bibliography

None.