Language Vulnerabilities Report: The Fortran Annex

David Muxworthy d.muxworthy @ bcs.org.uk 29 September 2011

TR 24772:2010

TR 24772 ”Guidance to Avoiding Vulnerabilities in Programming Languages through Language Selection and Use”

The initial project proposal in 2005 included:

- The purpose of this project is to prepare comparative guidance spanning a large number of programming languages, so that application developers will be better informed regarding the vulnerabilities inherent to candidate languages and the costs of avoiding such vulnerabilities
- The guidance could be applicable to any software development project applying the programming languages considered in the TR.
- Any programming language contains constructs that are vague or difficult to use.

Development history

2005 SC22 approved project by 10 votes to 2 (Netherlands and UK)

2006 'Other Working Group - Vulnerabilities' formed in SC22;

BSI panel formed

2008 OWG-V formally established as SC22/WG23

2009 Preliminary draft Technical Report issued for SC22 ballot

Vote: 7 yes – 7 yes with comments – 1 no (UK) – 8 abstain

Comments occupy 15 pages

Draft Fortran annex composed but not incorporated

2010 Technical Report (first edition without language-specific annexes)

published in June

Available free of charge from
http://standards.iso.org/ittf/PubliclyAvailableStandards/c041542_ISO_IEC_TR_24772_2010.zip

Structure of the TR – main clauses

1 - 4 Scope; Normative references; Terms and definitions; Basic Concepts

5 Vulnerability issues

- **5.1 incomplete or evolving language specifications**
- **5.2 human cognitive limitations**
- **5.3 lack of predictable execution**
- **5.4 lack of portability and interoperability**
- **5.5 inadequate language intrinsic support**
- **5.6 language features prone to erroneous use**

6 Programming Language Vulnerabilities

- **6.1 – 6.53 Different vulnerabilities [below]**

7 Application Vulnerabilities

Language Vulnerabilities

6.1 General
6.2 Obscure Language Features
6.3 Unspecified Behaviour
6.4 Undefined Behaviour
:
6.20 Unchecked Array Indexing
6.21 Unchecked Array Copying
:
6.33 Side-effects and Order of Evaluation
:
6.44 Recursion
:
6.53 Unanticipated Exceptions from Library Routines

Structure of the TR - annexes

A. Guideline Selection Process

B. Skeleton template for use in proposing programming language vulnerabilities

C. Skeleton template for use in proposing application vulnerabilities

D. Vulnerability Outline and List

E. Vulnerability descriptions for <language>

- **E.1 <language>.1 Identification of standards**
- **E.2 <language>.2 General terminology and concepts**
- **E.3 <language>.<x> <Vulnerability Name>**

Format of each Language-Specific Annex

For each language:

- Identification of standards
- General terminology and concepts

For each vulnerability within each language:

- Status and history
- Terminology and features
- Description of vulnerability
- Avoiding the vulnerability or mitigating its effects
- Implications for standardization
- Bibliography

Example of a Fortran vulnerability description (from 2009 draft)

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.3.18.2 and 3.18.3 expanded

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.3.2 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.3.5 Deprecated Language Features

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.

Language-specific annexes

Language-specific annexes are being developed, or are planned, for

- Ada
- C
- Fortran
- Python
- Ruby
- SPARK
- SQL
- possibly others

Consistency across languages? Example of “Obscure Language Features”

Ada

“Because some areas are specialized, it is likely that a programmer not versed in a special area might misuse features for that area.”

Fortran

Describes possible obscurities arising from use of redundant statements and features such as common blocks potentially going out of scope.

Python

“Python has some obscure language features as described below”

Ruby

“This vulnerability is not applicable to Ruby.”

Annex Development: A Moving Target

The PDTR issued for voting in 2009 had 48 language vulnerability subclauses

The draft Fortran annex, written in August 2009, has 51 vulnerability subclauses

The published TR has 53 vulnerability subclauses

The proposed second edition, as at July 2011, has 57 vulnerability subclauses

Language Vulnerability Differences, Fortran annex and TR

Not in Fortran annex but in TR:

- Namespace Issues

In Fortran annex and in TR but not in putative second edition:

- Choice of Filenames and other External Identifiers (moved to application vulnerabilities)
- Boundary Beginning Violation (subsumed into Unchecked Array Indexing)

New in second edition:

- Using Shift Operations for Multiplication and Division
- Dead Store
- Inter-language Calling
- Suppression of Language-defined Run-time Checking
- Provision of Inherently Unsafe Operations

Work to be done

1. Review entries for general introduction to Fortran annex
2. Review existing entries for Fortran language vulnerabilities
3. Produce entries for new vulnerabilities
4. Keep up to date with the moving target of the developing draft

References

WG23 website:
http://grouper.ieee.org/groups/plv/

n0220 draft Fortran annex (24 August 2009)
n0296 draft Ada annex
n0331 draft Ruby annex
n0347 draft Python annex
n0352 on-going working version of TR revision (July 2011)

and so on

To join the email group developing the Fortran annex, contact
Dan Nagle <dannagle @ verizon.net>