ISO/IEC JTC 1/SC 22/WG 23 N0192

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

ISO working group on Guidance for Avoiding Vulnerabilities through language selection and use

John Benito, Convener
Jim Moore, Secretary

June 2009

## The Problem

- ❑ Any programming language has constructs that are imperfectly defined, implementation dependent or difficult to use correctly.
- ❑ As a result, software programs sometimes execute differently than intended by the writer.
- ❑ In some cases, these vulnerabilities can be exploited by hostile parties.
  - – Can compromise safety, security and privacy.
  - – Can be used to make additional attacks.

## Complicating Factors

- The choice of programming language for a project is not solely a technical decision and is not made solely by software engineers.
- Some vulnerabilities cannot be mitigated by better use of the language but require mitigation by other methods, e.g. review, static analysis.

## WG 23 Vulnerability Product

- A type III Technical Report
  - A document containing information of a different kind from that which is normally published as an International Standard
  - The product will not contain *normative* statements, but information and suggestions
- Project is to work on a set of *common mode* failures that occur across a variety of languages
  - Not all vulnerabilities are common to all languages, that is, some manifest in just a language
- Annexes to the report will describe how the  vuln erabilities relate to specific languages.

## WG 23 Vulnerability Product

- No single programming language or family of programming languages is to be singled out
  - As many programming languages as possible should be involved
  - Need not be just the languages defined by ISO Standards

## Audience

- *Safety*: Products where it is critical to prevent behavior which might lead to human injury, and it is justified to spend additional development money
- *Security*: Products where it is critical to secure data or access, and it is justified to spend additional development money
- *Mission-Critical*: Products where it is important to prevent behaviour that can lead to losses
- *Modeling and Simulation*: Products which require unusual reliability because the cost of computation is high

## Approach to Identifying Vulnerabilities

- *Empirical approach:* Observe the vulnerabilities that occur in the wild and describe them, e.g. buffer overrun, execution of unvalidated remote content
- *Analytical approach:* Identify potential vulnerabilities through analysis of programming languages
  - This just might help in identifying tomorrows vulnerabilities.

## Measure of Success

- Provide guidance to users of programming languages that:
  - Assists them in improving the predictability of the execution of their software even in the presence of an attacker
  - Informs their selection of an appropriate programming language for their job
- Provide feedback to programming language standardization groups, resulting in the improvement of programming language standards.

## Progress

- Organization:
  - Project was originally assigned to a temporary group, an “other working group” called OWGV.
  - In September 2008, SC 22 created WG 23 to continue the work
- Meetings:
  - Email reflector, Wiki and Web site are used during and between meetings
  - Web conferencing is now being used to save travel
  - Ten meetings have been held hosted by six national bodies
  - Meetings are planned through 2009

## **Progress (**continued**)**

- Progress through standards process:
  - Working Group Level
    - Working Draft (WD) – several of them
  - Parent (SC 22) Level
    - PDTR registration
    - PDTR ballot repeated until consensus is obtained and the document is stable, two or three times is typical
  - Management (JTC 1) Level
    - DTR Ballot
  - Publication by ISO/IEC is currently planned for 2010

## **Progress (**continued**)**

- HOWEVER…
  - We need the assistance of language working groups to develop language-specific annexes

More information: http://aitc.aitcnet.org/isai

## Outline of current Draft

- ❑ Scope
- ❑ References
- ❑ Terms and Definitions
- ❑ Vulnerability Issues
- ❑ Programming Language Vulnerabilities
  (Currently 51 of them)
- ❑ Application Vulnerabilities
  (Currently 19 of them, selected because of relati
  onship to languages)

## Current Draft outline (continued)

- Annexes
  - A - Guideline Recommendation Factors
  - B - Guideline Selection Process
  - C - Skeleton template for use in proposing programming language vulnerabilities
  - D - Skeleton template for use in proposing application vulnerabilities
  - E - Vulnerability Outline
  - F - Skeleton template for use in proposing language specific information for vulnerabilities

## Vulnerability Template

- The body of Technical Report describes vulnerabilities in a generic manner, including:
  - Brief description of application vulnerability
  - Cross-reference to enumerations and other classifications
    - CWE (common weakness enumeration)
    - JSF AV Rules (Joint Strike Fighter, Air Vehicle)
    - MISRA C 2004
    - MISRA C++ 2008
    - CERT/CC guidelines
    - ISO/IEC TR 15942:2000
  - Description of failure mechanism, i.e. how coding problem relates to application vulnerability
  - Applicable language characteristics
  - Avoiding or mitigating the vulnerability
  - Implications for standardization
  - Bibliography

## Vulnerability Template (continued)

- Annexes will provide language-specific treatments of each vulnerability.

**Note:** Currently the languages working on this are

- Ada,
- Fortran, and
- C.

## Vulnerability Outline

- E.1. Human Factors
- E.2. Environment
- E.3. Core Language Issues
- E.4. Pre-processor
- E.5. Declarations and Definitions
- E.6. Types
- E.7. Templates/Generics
- E.8. Initialization
- E.9. Type Conversions/Limits

## **Vulnerability Outline (**continued**)**

- ❑ E.10. Operators/Expressions
- ❑ E.11. Control Flow
- ❑ E.12. Internal interfaces
- ❑ E.13. External interfaces
- ❑ E.14. Miscellaneous

# Description of a vulnerability

## 6.1 Obscure Language Features [BRS]

### 6.1.1 Description of application vulnerability

- Every programming language has features that are obscure, difficult to understand or difficult to use correctly. The problem is compounded if a software design must be reviewed by people who may not be language experts, such as, hardware engineers, human-factors engineers, or safety officers. Even if the design and code are initially correct, maintainers of the software may not fully understand the intent. The consequences of the problem are more severe if the software is to be used in trusted applications, such as safety or mission critical ones.
- Misunderstood language features or misunderstood code sequences can lead to application vulnerabilities in development or in maintenance.

## Description of a vulnerability (continued)

### 6.1.2 Cross reference

- JSF AV Rules: 84, 86, 88, and 97
- MISRA C 2004:  3.2, 10.2, 13.1, 20.6-20.12, 12.10, and 17.5
- MISRA C++ 2008: 0-2-1, 2-3-1, and 12-1-1
- CERT/CC guidelines: FIO03-C, MSC05-C, MSC30-C, and MSC31-C.
- ISO/IEC TR 15942:2000: 5.4.2, 5.6.2 and 5.9.3

## Description of a vulnerability (continued)

### 6.1.3 Mechanism of failure

- The use of obscure language features can lead to an application vulnerability in several ways:
  - The original programmer may misunderstand the correct usage of the feature and could utilize it incorrectly in the design or code it incorrectly.
  - Reviewers of the design and code may misunderstand the intent or the usage and overlook problems.
  - Maintainers of the code cannot fully understand the intent or the usage and could introduce problems during maintenance.

## Description of a vulnerability (continued)

### 6.1.4 Applicable language characteristics

- This vulnerability description is intended to be applicable to any language.

## **Description of a vulnerability** (continued)

### 6.1.5 Avoiding the vulnerability or mitigating its effects

- Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:
  - Individual programmers should avoid the use of language features that are obscure or difficult to use, especially in combination with other difficult language features. Organizations should adopt coding standards that discourage use of such features or show how to use them correctly.
  - Organizations developing software with critically important requirements should adopt a mechanism to monitor which language features are correlated with failures during the development process and during deployment.

## Continued …

- Organizations should adopt or develop stereotypical idioms for the use of difficult language features, codify them in organizational standards, and enforce them via review processes.
- Avoid the use of complicated features of a language.
- Avoid the use of rarely used constructs that could be difficult for entry-level maintenance personnel to understand.
- Static analysis can be used to find incorrect usage of some language features.
- It should be noted that consistency in coding is desirable for each of review and maintenance. Therefore, the desirability of the particular alternatives chosen for inclusion in a coding standard does not need to be empirically proven.

## Description of a vulnerability (continued)

### 6.1.6 Implications for standardization

- In future standardization activates, the following items should be considered:
  - Language designers should consider removing or deprecating obscure, difficult to understand, or difficult to use features.
  - Language designers should provide language directives that optionally disable obscure language features.

## **Description of a vulnerability** (continued)

**6.1.7 Bibliography**

- Hatton 17: Use of obscure language features

## Discussion

- Should not go to ballot without language-specific annexes because the presence of the annexes would change the main document:
  - Examples might move to annexes.
  - There may be resistance to changing the main document after it has been successfully balloted.
  - Adding annexes after the initial TR is approved would give the impression of instability.
- Doing prototype of language-specific annexes
  - Select a subset of descriptions and write a sample annex
  - To experiment with formats
  - To look at what kind of changes to the main document would be appropriate
- WG 23 has added Annex F as a template for language-specific annex.
- New PDTR text should be available on the WG 23 web site.