TECHNICAL REPORT

ISO/IEC TR 24772

First edition
2010-10-01

Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages through language selection and use

Technologies de l'information — Langages de programmation — Conduite pour éviter les vulnérabilités dans les langages de programmation à travers la sélection et l'usage de la langue

Reference number
ISO/IEC TR 24772:2010(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example), it may decide to publish a Technical Report. A Technical Report is entirely informative in nature and shall be subject to review every five years in the same manner as an International Standard.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 24772 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces*.

Introduction

All programming languages contain constructs that are incompletely specified, exhibit undefined behaviour, are implementation-dependent, or are difficult to use correctly. The use of those constructs may therefore give rise to *vulnerabilities*, as a result of which, software programs can execute differently than intended by the writer. In some cases, these vulnerabilities can compromise the safety of a system or be exploited by attackers to compromise the security or privacy of a system.

This Technical Report is intended to provide guidance spanning multiple programming languages, so that application developers will be better able to avoid the programming constructs that lead to vulnerabilities in software written in their chosen language and their attendant consequences. This guidance can also be used by developers to select source code evaluation tools that can discover and eliminate some constructs that could lead to vulnerabilities in their software or to select a programming language that avoids anticipated problems.

It should be noted that this Technical Report is inherently incomplete. It is not possible to provide a complete list of programming language vulnerabilities because new weaknesses are discovered continually. Any such report can only describe those that have been found, characterized, and determined to have sufficient probability and consequence.

Furthermore, to focus its limited resources, the working group developing this report decided to defer comprehensive treatment of several subject areas until future editions of the report. These subject areas include:

- Object-oriented language features (Although some simple issues related to inheritance are described in RIP)
- Concurrency
- Numerical analysis (although some simple items regarding the use of floating point are described in PLF)
- Scripting languages
- Inter-language operability
- Language-specific annexes

Information technology — Programming languages — Guidance to avoiding vulnerabilities in programming languages through language selection and use

1 Scope

This Technical Report specifies software programming language vulnerabilities to be avoided in the development of systems where assured behaviour is required for security, safety, mission critical and business critical software. In general, this guidance is applicable to the software developed, reviewed, or maintained for any application.

Vulnerabilities are described in a generic manner that is applicable to a broad range of programming languages.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 80000–2:2009, *Quantities and units — Part 2: Mathematical signs and symbols to be use in the natural sciences and technology*

ISO/IEC 2382–1:1993, *Information technology — Vocabulary — Part 1: Fundamental terms*

3 Terms and definitions, symbols and conventions

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 2382–1 and the following apply. Other terms are defined where they appear in *italic* type.

3.1.1
language vulnerability
property (of a programming language) that can contribute to, or that is strongly correlated with, application vulnerabilities in programs written in that language

> **Note 1:** The term "property" can mean the presence or the absence of a specific feature, used singly or in combination. As an example of the absence of a feature, encapsulation (control of where names can be referenced from) is generally considered beneficial since it narrows the interface between modules and can help prevent data corruption. The absence of encapsulation from a programming language can thus be regarded as a vulnerability. Note that a property together with its complement can both be considered language vulnerabilities. For example, automatic storage reclamation (garbage collection) can be a

vulnerability since it can interfere with time predictability and result in a safety hazard. On the other hand, the absence of automatic storage reclamation can also be a vulnerability since programmers can mistakenly free storage prematurely, resulting in dangling references.

3.1.2

application vulnerability

security vulnerability or safety hazard, or defect

3.1.3

security vulnerability

weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat

3.1.4

safety hazard

potential source of harm

Note 1: IEC 61508–4: defines a “Hazard” as a “potential source of harm”, where “harm” is “physical injury or damage to the health of people either directly or indirectly as a result of damage to property or to the environment”.

Note 2: IEC 61508 is titled “Functional safety of electrical/electronic/ programmable electronic safety-related systems”, with part 4 being “Definitions and abbreviations”. Hence within IEC 61508 the “safety” context of “safety hazard” is assumed.

Note 3: IEC 61508 cites ISO/IEC Guide 51 as the source for the definition.

Note 4: Some derived standards, such as UK Defence Standard 00-56, broaden the definition of “harm” to include material and environmental damage (not just harm to people caused by property and environmental damage).

3.1.5

safety-critical software

software for applications where failure can cause very serious consequences such as human injury or death

Note 1: IEC 61508–4: defines “Safety-related software” as “software that is used to implement safety functions in a safety-related system.

Note 2: Notwithstanding that in some domains a distinction is made between safety-related (can lead to any harm) and safety-critical (life threatening), this Technical Report uses the term *safety-critical* for all vulnerabilities that can result in safety hazards.

3.1.6

software quality

degree to which software implements the requirements described by its specification and the degree to which the characteristics of a software product fulfil its requirements

3.1.7

predictable execution

property of the program such that all possible executions have results that can be predicted from the source code

Note 1: All the relevant language-defined implementation characteristics and knowledge of the universe of execution must be known.

Note 2: In some environments, this would raise issues regarding numerical stability, exceptional processing, and concurrent execution.

Note 3: Predictable execution is an ideal that must be approached keeping in mind the limits of human capability, knowledge, availability of tools, and other factors. Neither this Technical Report nor any standard ensures predictable execution. Rather this Technical Report provides advice on improving predictability. The purpose of this Technical Report is to assist a reasonably competent programmer to approach the ideal of predictable execution.

Note 4: The following terms are used in relation to "Predictable execution".

- **Unspecified behaviour:** A situation where the implementation of a language will have to make some choice from a finite set of alternatives, but that choice is not in general predictable by the programmer, for example, the order in which sub-expressions are evaluated in an expression in many languages.
- **Implementation-defined behaviour:** A situation where the implementation of a language will have to make some choice, and it is required that this choice be documented and available to the programmer.
- **Undefined behaviour:** A situation where the definition of a language can give no indication of what behaviour to expect from a program – it can be some form of catastrophic failure (a 'crash') or continued execution with some arbitrary data.

Note 5: This Technical Report includes a clause on **Unspecified functionality**. This notion is related to neither unspecified behaviour, which is a characteristic of an application, nor the language used to develop the application.

3.2 Symbols and conventions

3.2.1 Symbols

For the purposes of this document, the symbols given in ISO/IEC 80000–2 apply. Other symbols are defined where they appear in this document.

3.2.2 Conventions

Programming language token and syntactic token appear in `courier` font.

4 Basic Concepts

4.1 Not in Scope

This Technical Report does not address software engineering and management issues such as how to design and implement programs, use configuration management tools, use managerial processes, and perform process improvement. Furthermore, the specification of properties to be assured is not treated.

The specification of an application is *not* within the scope.

While this Technical Report does not discuss specification or design issues, there is recognition that boundaries among the various activities are not clear-cut. This Technical Report seeks to avoid the debate about where low-level design ends and implementation begins by treating selected issues that some might consider design issues rather than coding issues.

4.2 Approach

Guidelines based on this Technical Report are likely to be highly leveraged in that they are likely to affect many times more people than the number that worked on them. Therefore guidelines such as these have the potential to make large savings, for a small cost, or to generate large unnecessary costs, for little benefit. For this reason, the writers of this Technical Report have taken a cautious approach to identifying programming language vulnerabilities. New vulnerability descriptions can be added over time, as experience and evidence are accumulated.

4.3 Intended Audience

The intended audience for this Technical Report is those who are concerned with assuring the existence of a critical property in the software of their system; that is, those who are developing, qualifying, or maintaining a software system and need to avoid language constructs that could cause the software to execute in a manner other than intended.

As described in the following paragraphs, developers of applications that have clear safety, security or mission criticality are expected to be aware of the risks associated with their code and could use this Technical Report to ensure that their development practices address the issues presented by the chosen programming languages, for example by subsetting or providing coding guidelines.

That should not be taken to mean that other developers can ignore this Technical Report. A weakness in an application that of itself has no direct criticality may provide the route by which an attacker gains control of a system or may otherwise disrupt co-hosted applications that are safety, security or mission critical.

It is hoped that such developers would use this Technical Report to ensure that common vulnerabilities are removed or at least minimized from all applications.

Several specific audiences for this International Technical Report have been identified and are described below.

4.3.1 Safety-Critical Applications

Users who may benefit from this Technical Report include those developing, qualifying, or maintaining a system where it is critical to prevent behaviour that might lead to:

- loss of human life or human injury, or
- damage to the environment.

4.3.2 Security-Critical Applications

Users who may benefit from this Technical Report include those developing, qualifying, or maintaining a system where it is critical to exhibit security properties of:

- confidentiality,
- integrity, and
- availability.

4.3.3 Mission-Critical Applications

Users who may benefit from this Technical Report include those developing, qualifying, or maintaining a system where it is critical to prevent behaviour that might lead to:

- property loss or damage, or
- economic loss or damage.

4.3.4 Business-Critical Systems

Users who may benefit from this Technical Report include those developing, qualifying, or maintaining a system whose correctness and integrity are essential for successful operation of a business or enterprise.

4.3.5 Modeling and Simulation Applications

People who may benefit from this Technical Report include those who are primarily experts in areas other than programming but need to use computation as part of their work. Such people include scientists, engineers, economists, and statisticians. They require high confidence in the applications they write and use because of the increasing complexity of the calculations made (and the consequent use of teams of programmers each contributing expertise in a portion of the calculation), or to the costs of invalid results, or to the expense of individual calculations implied by a very large number of processors used and/or very long execution times needed to complete the calculations. These circumstances give a consequent need for high reliability and motivate the need felt by these programmers for the guidance offered in this Technical Report.

4.4 How to Use This Document

This Technical Report gathers language-independent descriptions of programming language vulnerabilities, as well as selected application vulnerabilities, which have occurred in the past and are likely to occur again. Because new vulnerabilities are always being discovered, it is anticipated that this Technical Report will be revised and new descriptions added. For that reason, a scheme that is distinct from Technical Report sub-clause numbering

has been adopted to identify the vulnerability descriptions. Each description has been assigned an arbitrarily generated, unique three-letter code. These codes should be used in preference to sub-clause numbers when referencing descriptions.

The main part of this Technical Report contains descriptions that are intended to be language-independent to the greatest possible extent. Future editions will include annexes that apply the generic guidance to particular programming languages.

This Technical Report has been written with several possible usages in mind:

- Programmers familiar with the vulnerabilities of a specific language can reference the guide for more generic descriptions and their manifestations in less familiar languages.
- Tool vendors can use the three-letter codes as a succinct way to “profile” the selection of vulnerabilities considered by their tools.
- Individual organizations may wish to write their own coding standards intended to reduce the number of vulnerabilities in their software products. The guide can assist in the selection of vulnerabilities to be addressed in those standards and the selection of coding guidelines to be enforced.
- Organizations or individuals selecting a language for use in a project may want to consider the vulnerabilities inherent in various candidate languages.

The following clauses include suggestions for ways of avoiding the vulnerabilities. It should be noted that these include techniques that can be applied during development, and those that must be implemented as run-time checks. The former are likely to be appropriate to all applications. For some applications, it is relatively more important to ensure that potential run-time errors are eliminated during development because there may be insufficient opportunity to recover from them. For long-running simulations, run-time checks may increase the run-time to the point where the prediction loses value, or to the point where the cost of calculation becomes prohibitive given the value of the simulation results. Source code checking tools can be used to automatically enforce some coding rules and standards.

Clause 2 provides Normative references, and Clause 3 provides Terms, definitions, symbols and conventions.

Clause 4 provides the basic concepts used for this Technical Report.

Clause 5, *Vulnerability Issues*, provides rationale for this Technical Report and explains how many of the vulnerabilities occur.

Clause 6, *Programming Language Vulnerabilities*, provides language-independent descriptions of vulnerabilities in programming languages that can lead to application vulnerabilities. Each description provides:

- a summary of the vulnerability,
- characteristics of languages where the vulnerability may be found,
- typical mechanisms of failure,
- techniques that programmers can use to avoid the vulnerability, and
- ways that language designers can modify language specifications in the future to help programmers mitigate the vulnerability.

Clause 7, *Application Vulnerabilities*, provides descriptions of selected application vulnerabilities which have been found and exploited in a number of applications and which have well known mitigation techniques, and which result from design decisions made by coders in the absence of suitable language library routines or other mechanisms. For these vulnerabilities, each description provides:

- a summary of the vulnerability,
- typical mechanisms of failure, and
- techniques that programmers can use to avoid the vulnerability.

Annexes A through C are templates and guidelines that were used in the identification, selection, and creation of the vulnerabilities that were generated for clauses 6 and 7. They can be used to guide the generation of future language vulnerabilities and application vulnerabilities.

Annex D, *Vulnerability Outline and List*, is a categorization of the vulnerabilities of this report in the form of a hierarchical outline and a list of the vulnerabilities arranged in alphabetic order by their three letter code.

Annex E, *Language Specific Vulnerability Template*, is a template for the writing of programming language specific annexes that explain how the vulnerabilities from clause 6 are realized in that programming language (or show how they are absent), and how they might be mitigated in language-specific terms. Future revisions of this Technical Report are planned to contain language-specific annexes that are developed using Annex E.