TECHNICAL REPORT

ISO/IEC TR 24772-1

First edition
2019-12

Programming languages — Guidance to avoiding vulnerabilities in programming languages —

Part 1: Language-independent guidance

Langages de programmation — Conduite pour éviter les vulnérabilités dans les langages de programmation —

Partie 1: Conduite indépendante du langage

Reference number
ISO/IEC TR 24772-1:2019(E)

© ISO/IEC 2019



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword ... **xv**

Introduction ... **xvi**

1 Scope ... **1**

2 Normative references ... **1**

3 Terms and definitions ... **1**
3.1 Terms related to communication ... 1
3.2 Terms related to execution model ... 2
3.3 Properties ... 4
3.4 Safety ... 4
3.5 Vulnerabilities ... 4

4 Applying this document ... **5**

5 Vulnerability issues and general avoidance mechanisms ... **7**
5.1 Predictable execution ... 7
5.2 Sources of unpredictability in language specification ... 8
5.2.1 Incomplete or evolving specification ... 8
5.2.2 Undefined behaviour ... 8
5.2.3 Unspecified behaviour ... 8
5.2.4 Implementation-defined behaviour ... 8
5.2.5 Difficult features ... 8
5.2.6 Inadequate language support ... 8
5.3 Sources of unpredictability in language usage ... 9
5.3.1 Porting and interoperation ... 9
5.3.2 Compiler selection and usage ... 9
5.4 Top avoidance mechanisms ... 9

6 Programming language vulnerabilities ... **11**
6.1 General ... 11
6.2 Type system [IHN] ... 11
6.2.1 Description of application vulnerability ... 11
6.2.2 Cross reference ... 11
6.2.3 Mechanism of failure ... 11
6.2.4 Applicable language characteristics ... 13
6.2.5 Avoiding the vulnerability or mitigating its effects ... 13
6.2.6 Implications for language design and evolution ... 14
6.3 Bit representations [STR] ... 14
6.3.1 Description of application vulnerability ... 14
6.3.2 Cross reference ... 14
6.3.3 Mechanism of failure ... 14
6.3.4 Applicable language characteristics ... 15
6.3.5 Avoiding the vulnerability or mitigating its effects ... 15
6.3.6 Implications for language design and evolution ... 15
6.4 Floating-point arithmetic [PLF] ... 15
6.4.1 Description of application vulnerability ... 15
6.4.2 Cross reference ... 16
6.4.3 Mechanism of failure ... 16
6.4.4 Applicable language characteristics ... 17
6.4.5 Avoiding the vulnerability or mitigating its effects ... 17
6.4.6 Implications for language design and evolution ... 17
6.5 Enumerator issues [CCB] ... 18
6.5.1 Description of application vulnerability ... 18
6.5.2 Cross reference ... 18
6.5.3 Mechanism of failure ... 18
6.5.4 Applicable language characteristics ... 19

© ISO/IEC 2019 – All rights reserved

6.5.5 Avoiding the vulnerability or mitigating its effects 19
6.5.6 Implications for language design and evolution 19
6.6 Conversion errors [FLC] 19
6.6.1 Description of application vulnerability 19
6.6.2 Cross reference 20
6.6.3 Mechanism of failure 20
6.6.4 Applicable language characteristics 20
6.6.5 Avoiding the vulnerability or mitigating its effects 21
6.6.6 Implications for language design and evolution 21
6.7 String termination [CJM] 21
6.7.1 Description of application vulnerability 21
6.7.2 Cross reference 21
6.7.3 Mechanism of failure 22
6.7.4 Applicable language characteristics 22
6.7.5 Avoiding the vulnerability or mitigating its effects 22
6.7.6 Implications for language design and evolution 22
6.8 Buffer boundary violation (buffer overflow) [HCB] 22
6.8.1 Description of application vulnerability 22
6.8.2 Cross reference 22
6.8.3 Mechanism of failure 23
6.8.4 Applicable language characteristics 23
6.8.5 Avoiding the vulnerability or mitigating its effects 24
6.8.6 Implications for language design and evolution 24
6.9 Unchecked array indexing [XYZ] 24
6.9.1 Description of application vulnerability 24
6.9.2 Cross reference 25
6.9.3 Mechanism of failure 25
6.9.4 Applicable language characteristics 25
6.9.5 Avoiding the vulnerability or mitigating its effects 26
6.9.6 Implications for language designers 26
6.10 Unchecked array copying [XYW] 26
6.10.1 Description of application vulnerability 26
6.10.2 Cross reference 26
6.10.3 Mechanism of failure 26
6.10.4 Applicable language characteristics 27
6.10.5 Avoiding the vulnerability or mitigating its effects 27
6.10.6 Implications for language design and evolution 27
6.11 Pointer type conversions [HFC] 27
6.11.1 Description of application vulnerability 27
6.11.2 Cross reference 28
6.11.3 Mechanism of failure 28
6.11.4 Applicable language characteristics 28
6.11.5 Avoiding the vulnerability or mitigating its effects 28
6.11.6 Implications for language design and evolution 29
6.12 Pointer arithmetic [RVG] 29
6.12.1 Description of application vulnerability 29
6.12.2 Cross reference 29
6.12.3 Mechanism of failure 29
6.12.4 Applicable language characteristics 29
6.12.5 Avoiding the vulnerability or mitigating its effects 29
6.12.6 Implications for language design and evolution 29
6.13 Null pointer dereference [XYH] 29
6.13.1 Description of application vulnerability 29
6.13.2 Cross reference 30
6.13.3 Mechanism of failure 30
6.13.4 Applicable language characteristics 30
6.13.5 Avoiding the vulnerability or mitigating its effects 30
6.13.6 Implications for language design and evolution 30

 © ISO/IEC 2019 – All rights reserved

6.14 Dangling reference to heap [XYK] 30
6.14.1 Description of application vulnerability 30
6.14.2 Cross reference 31
6.14.3 Mechanism of failure 31
6.14.4 Applicable language characteristics 31
6.14.5 Avoiding the vulnerability or mitigating its effects 32
6.14.6 Implications for language design and evolution 32
6.15 Arithmetic wrap-around error [FIF] 32
6.15.1 Description of application vulnerability 32
6.15.2 Cross reference 32
6.15.3 Mechanism of failure 33
6.15.4 Applicable language characteristics 33
6.15.5 Avoiding the vulnerability or mitigating its effects 33
6.15.6 Implications for language design and evolution 33
6.16 Using shift operations for multiplication and division [PIK] 34
6.16.1 Description of application vulnerability 34
6.16.2 Cross reference 34
6.16.3 Mechanism of failure 34
6.16.4 Applicable language characteristics 34
6.16.5 Avoiding the vulnerability or mitigating its effects 34
6.16.6 Implications for language design and evolution 34
6.17 Choice of clear names [NAI] 35
6.17.1 Description of application vulnerability 35
6.17.2 Cross reference 35
6.17.3 Mechanism of Failure 35
6.17.4 Applicable language characteristics 36
6.17.5 Avoiding the vulnerability or mitigating its effects 36
6.17.6 Implications for language design and evolution 36
6.18 Dead store [WXQ] 36
6.18.1 Description of application vulnerability 36
6.18.2 Cross reference 36
6.18.3 Mechanism of failure 37
6.18.4 Applicable language characteristics 37
6.18.5 Avoiding the vulnerability or mitigating its effects 37
6.18.6 Implications for language design and evolution 37
6.19 Unused variable [YZS] 38
6.19.1 Description of application vulnerability 38
6.19.2 Cross reference 38
6.19.3 Mechanism of failure 38
6.19.4 Applicable language characteristics 38
6.19.5 Avoiding the vulnerability or mitigating its effects 38
6.19.6 Implications for language design and evolution 38
6.20 Identifier name reuse [YOW] 38
6.20.1 Description of application vulnerability 38
6.20.2 Cross reference 39
6.20.3 Mechanism of failure 39
6.20.4 Applicable language characteristics 40
6.20.5 Avoiding the vulnerability or mitigating its effects 40
6.20.6 Implications for language design and evolution 40
6.21 Namespace issues [BJL] 40
6.21.1 Description of application vulnerability 40
6.21.2 Cross-references 41
6.21.3 Mechanism of failure 41
6.21.4 Applicable language characteristics 41
6.21.5 Avoiding the vulnerability or mitigating its effects 41
6.21.6 Implications for language design and evolution 42
6.22 Initialization of variables [LAV] 42
6.22.1 Description of application vulnerability 42

© ISO/IEC 2019 – All rights reserved

6.22.2 Cross reference 42
6.22.3 Mechanism of failure 42
6.22.4 Applicable language characteristics 43
6.22.5 Avoiding the vulnerability or mitigating its effects 43
6.22.6 Implications for language design and evolution 44
6.23 Operator precedence and associativity [JCW] 44
6.23.1 Description of application vulnerability 44
6.23.2 Cross reference 44
6.23.3 Mechanism of failure 44
6.23.4 Applicable language characteristics 45
6.23.5 Avoiding the vulnerability or mitigating its effects 45
6.23.6 Implications for language design and evolution 45
6.24 Side-effects and order of evaluation of operands [SAM] 45
6.24.1 Description of application vulnerability 45
6.24.2 Cross reference 45
6.24.3 Mechanism of failure 46
6.24.4 Applicable language characteristics 46
6.24.5 Avoiding the vulnerability or mitigating its effects 46
6.24.6 Implications for language design and evolution 46
6.25 Likely incorrect expression [KOA] 47
6.25.1 Description of application vulnerability 47
6.25.2 Cross reference 47
6.25.3 Mechanism of failure 47
6.25.4 Applicable language characteristics 48
6.25.5 Avoiding the vulnerability or mitigating its effects 48
6.25.6 Implications for language design and evolution 48
6.26 Dead and deactivated code [XYQ] 48
6.26.1 Description of application vulnerability 48
6.26.2 Cross reference 48
6.26.3 Mechanism of failure 49
6.26.4 Applicable language characteristics 50
6.26.5 Avoiding the vulnerability or mitigating its effects 50
6.26.6 Implications for language design and evolution 50
6.27 Switch statements and static analysis [CLL] 50
6.27.1 Description of application vulnerability 50
6.27.2 Cross reference 50
6.27.3 Mechanism of failure 51
6.27.4 Applicable language characteristics 51
6.27.5 Avoiding the vulnerability or mitigating its effects 51
6.27.6 Implications for language design and evolution 51
6.28 Demarcation of control flow [EOJ] 52
6.28.1 Description of application vulnerability 52
6.28.2 Cross reference 52
6.28.3 Mechanism of failure 52
6.28.4 Applicable language characteristics 52
6.28.5 Avoiding the vulnerability or mitigating its effects 52
6.28.6 Implications for language design and evolution 53
6.29 Loop control variables [TEX] 53
6.29.1 Description of application vulnerability 53
6.29.2 Cross reference 53
6.29.3 Mechanism of failure 53
6.29.4 Applicable language characteristics 53
6.29.5 Avoiding the vulnerability or mitigating its effects 53
6.29.6 Implications for language design and evolution 54
6.30 Off-by-one error [XZH] 54
6.30.1 Description of application vulnerability 54
6.30.2 Cross reference 54
6.30.3 Mechanism of failure 54

 © ISO/IEC 2019 – All rights reserved

6.30.4 Applicable language characteristics 55
6.30.5 Avoiding the vulnerability or mitigating its effects 55
6.30.6 Implications for language design and evolution 55
6.31 Unstructured programming [EWD] 55
6.31.1 Description of application vulnerability 55
6.31.2 Cross reference 55
6.31.3 Mechanism of failure 56
6.31.4 Applicable language characteristics 56
6.31.5 Avoiding the vulnerability or mitigating its effects 56
6.31.6 Implications for language design and evolution 56
6.32 Passing parameters and return values [CSJ] 56
6.32.1 Description of application vulnerability 56
6.32.2 Cross reference 57
6.32.3 Mechanism of failure 57
6.32.4 Applicable language characteristics 58
6.32.5 Avoiding the vulnerability or mitigating its effects 58
6.32.6 Implications for language design and evolution 58
6.33 Dangling references to stack frames [DCM] 58
6.33.1 Description of application vulnerability 58
6.33.2 Cross reference 59
6.33.3 Mechanism of failure 59
6.33.4 Applicable language characteristics 60
6.33.5 Avoiding the vulnerability or mitigating its effects 60
6.33.6 Implications for language design and evolution 60
6.34 Subprogram signature mismatch [OTR] 60
6.34.1 Description of application vulnerability 60
6.34.2 Cross reference 60
6.34.3 Mechanism of failure 61
6.34.4 Applicable language characteristics 61
6.34.5 Avoiding the vulnerability or mitigating its effects 61
6.34.6 Implications for language design and evolution 61
6.35 Recursion [GDL] 62
6.35.1 Description of application vulnerability 62
6.35.2 Cross reference 62
6.35.3 Mechanism of failure 62
6.35.4 Applicable language characteristics 62
6.35.5 Avoiding the vulnerability or mitigating its effects 62
6.35.6 Implications for language design and evolution 63
6.36 Ignored error status and unhandled exceptions [OYB] 63
6.36.1 Description of application vulnerability 63
6.36.2 Cross reference 63
6.36.3 Mechanism of failure 63
6.36.4 Applicable language characteristics 64
6.36.5 Avoiding the vulnerability or mitigating its effects 64
6.36.6 Implications for language design and evolution 65
6.37 Type-breaking reinterpretation of data [AMV] 65
6.37.1 Description of application vulnerability 65
6.37.2 Cross reference 65
6.37.3 Mechanism of failure 65
6.37.4 Applicable language characteristics 66
6.37.5 Avoiding the vulnerability or mitigating its effects 66
6.37.6 Implications for language design and evolution 66
6.38 Deep vs. shallow copying [YAN] 67
6.38.1 Description of application vulnerability 67
6.38.2 Cross reference 67
6.38.3 Mechanism of failure 67
6.38.4 Applicable language characteristics 67
6.38.5 Avoiding the vulnerability or mitigating its effects 67

© ISO/IEC 2019 – All rights reserved

6.38.6 Implications for language design and evolution 68
6.39 Memory leaks and heap fragmentation [XYL] 68
6.39.1 Description of application vulnerability 68
6.39.2 Cross reference 68
6.39.3 Mechanism of failure 68
6.39.4 Applicable language characteristics 68
6.39.5 Avoiding the vulnerability or mitigating its effects 69
6.39.6 Implications for language design and evolution 69
6.40 Templates and generics [SYM] 69
6.40.1 Description of application vulnerability 69
6.40.2 Cross reference 70
6.40.3 Mechanism of failure 70
6.40.4 Applicable language characteristics 70
6.40.5 Avoiding the vulnerability or mitigating its effects 71
6.40.6 Implications for language design and evolution 71
6.41 Inheritance [RIP] 71
6.41.1 Description of application vulnerability 71
6.41.2 Cross reference 71
6.41.3 Mechanism of failure 72
6.41.4 Applicable language characteristics 72
6.41.5 Avoiding the vulnerability or mitigating its effects 72
6.41.6 Implications for language design and evolution 73
6.42 Violations of the Liskov substitution principle or the contract model [BLP] 73
6.42.1 Description of application vulnerability 73
6.42.2 Cross reference 74
6.42.3 Mechanism of failure 74
6.42.4 Applicable language characteristics 74
6.42.5 Avoiding the vulnerability or mitigating its effects 74
6.42.6 Implications for language design and evolution 74
6.43 Redispatching [PPH] 74
6.43.1 Description of application vulnerability 74
6.43.2 Cross reference 75
6.43.3 Mechanism of failure 75
6.43.4 Applicable language characteristics 75
6.43.5 Avoiding the vulnerability or mitigating its effects 75
6.43.6 Implications for language design and evolution 75
6.44 Polymorphic variables [BKK] 75
6.44.1 Description of application vulnerability 75
6.44.2 Cross reference 76
6.44.3 Mechanism of failure 76
6.44.4 Applicable language characteristics 77
6.44.5 Avoiding the vulnerability or mitigating its effects 77
6.44.6 Implications for language design and evolution 77
6.45 Extra intrinsics [LRM] 77
6.45.1 Description of application vulnerability 77
6.45.2 Cross reference 77
6.45.3 Mechanism of failure 77
6.45.4 Applicable language characteristics 78
6.45.5 Avoiding the vulnerability or mitigating its effects 78
6.45.6 Implications for language design and evolution 78
6.46 Argument passing to library functions [TRJ] 78
6.46.1 Description of application vulnerability 78
6.46.2 Cross reference 78
6.46.3 Mechanism of failure 78
6.46.4 Applicable language characteristics 79
6.46.5 Avoiding the vulnerability or mitigating its effects 79
6.46.6 Implications for language design and evolution 79
6.47 Inter-language calling [DJS] 79

© ISO/IEC 2019 – All rights reserved

6.47.1 Description of application vulnerability 79
6.47.2 Cross reference 79
6.47.3 Mechanism of failure 79
6.47.4 Applicable language characteristics 80
6.47.5 Avoiding the vulnerability or mitigating its effects 80
6.47.6 Implications for language design and evolution 81
6.48 Dynamically-linked code and self-modifying code [NYY] 81
6.48.1 Description of application vulnerability 81
6.48.2 Cross reference 81
6.48.3 Mechanism of failure 81
6.48.4 Applicable language characteristics 81
6.48.5 Avoiding the vulnerability or mitigating its effects 82
6.48.6 Implications for language design and evolution 82
6.49 Library signature [NSQ] 82
6.49.1 Description of application vulnerability 82
6.49.2 Cross reference 82
6.49.3 Mechanism of failure 82
6.49.4 Applicable language characteristics 83
6.49.5 Avoiding the vulnerability or mitigating its effects 83
6.49.6 Implications for language design and evolution 83
6.50 Unanticipated exceptions from library routines [HJW] 83
6.50.1 Description of application vulnerability 83
6.50.2 Cross reference 83
6.50.3 Mechanism of failure 83
6.50.4 Applicable language characteristics 84
6.50.5 Avoiding the vulnerability or mitigating its effects 84
6.50.6 Implications for language design and evolution 84
6.51 Pre-processor directives [NMP] 84
6.51.1 Description of application vulnerability 84
6.51.2 Cross reference 84
6.51.3 Mechanism of failure 85
6.51.4 Applicable language characteristics 85
6.51.5 Avoiding the vulnerability or mitigating its effects 85
6.51.6 Implications for language design and evolution 85
6.52 Suppression of language-defined run-time checking [MXB] 86
6.52.1 Description of application vulnerability 86
6.52.2 Cross reference 86
6.52.3 Mechanism of Failure 86
6.52.4 Applicable language characteristics 86
6.52.5 Avoiding the vulnerability 86
6.52.6 Implications for language design and evolution 86
6.53 Provision of inherently unsafe operations [SKL] 87
6.53.1 Description of application vulnerability 87
6.53.2 Cross reference 87
6.53.3 Mechanism of Failure 87
6.53.4 Applicable language characteristics 87
6.53.5 Avoiding the vulnerability 87
6.53.6 Implications for language design and evolution 88
6.54 Obscure language features [BRS] 88
6.54.1 Description of application vulnerability 88
6.54.2 Cross reference 88
6.54.3 Mechanism of failure 88
6.54.4 Applicable language characteristics 88
6.54.5 Avoiding the vulnerability or mitigating its effects 88
6.54.6 Implications for language design and evolution 89
6.55 Unspecified behaviour [BQF] 89
6.55.1 Description of application vulnerability 89
6.55.2 Cross reference 89

© ISO/IEC 2019 – All rights reserved

6.55.3 Mechanism of failure ... 89
6.55.4 Applicable language characteristics ... 90
6.55.5 Avoiding the vulnerability or mitigating its effects ... 90
6.55.6 Implications for language design and evolution ... 90
6.56 Undefined behaviour [EWF] ... 90
6.56.1 Description of application vulnerability ... 90
6.56.2 Cross reference ... 91
6.56.3 Mechanism of failure ... 91
6.56.4 Applicable language characteristics ... 91
6.56.5 Avoiding the vulnerability or mitigating its effects ... 91
6.56.6 Implications for language design and evolution ... 92
6.57 Implementation-defined behaviour [FAB] ... 92
6.57.1 Description of application vulnerability ... 92
6.57.2 Cross reference ... 92
6.57.3 Mechanism of failure ... 92
6.57.4 Applicable language characteristics ... 93
6.57.5 Avoiding the vulnerability or mitigating its effects ... 93
6.57.6 Implications for language design and evolution ... 93
6.58 Deprecated language features [MEM] ... 93
6.58.1 Description of application vulnerability ... 93
6.58.2 Cross reference ... 94
6.58.3 Mechanism of failure ... 94
6.58.4 Applicable language characteristics ... 94
6.58.5 Avoiding the vulnerability or mitigating its effects ... 94
6.58.6 Implications for language design and evolution ... 95
6.59 Concurrency — Activation [CGA] ... 95
6.59.1 Description of application vulnerability ... 95
6.59.2 Cross-references ... 95
6.59.3 Mechanism of Failure ... 95
6.59.4 Applicable language characteristics ... 96
6.59.5 Avoiding the vulnerability or mitigating its effects ... 96
6.59.6 Implications for language design and evolution ... 96
6.60 Concurrency — Directed termination [CGT] ... 96
6.60.1 Description of application vulnerability ... 96
6.60.2 Cross-references ... 97
6.60.3 Mechanism of failure ... 97
6.60.4 Applicable language characteristics ... 97
6.60.5 Avoiding the vulnerability or mitigating its effect ... 97
6.60.6 Implications for language design and evolution ... 98
6.61 Concurrent data access [CGX] ... 98
6.61.1 Description of application vulnerability ... 98
6.61.2 Cross-references ... 98
6.61.3 Mechanism of failure ... 98
6.61.4 Applicable language characteristics ... 98
6.61.5 Avoiding the vulnerability or mitigating its effect ... 99
6.61.6 Implications for language design and evolution ... 99
6.62 Concurrency — Premature termination [CGS] ... 99
6.62.1 Description of application vulnerability ... 99
6.62.2 Cross-references ... 99
6.62.3 Mechanism of failure ... 100
6.62.4 Applicable language characteristics ... 100
6.62.5 Avoiding the vulnerability or mitigating its effect ... 100
6.62.6 Implications for language design and evolution ... 101
6.63 Lock protocol errors [CGM] ... 101
6.63.1 Description of application vulnerability ... 101
6.63.2 Cross-references ... 101
6.63.3 Mechanism of failure ... 102
6.63.4 Applicable language characteristics ... 102

 © ISO/IEC 2019 – All rights reserved

6.63.5 Avoiding the vulnerability or mitigating its effect 102
6.63.6 Implications for language design and evolution 103
6.64 Reliance on external format strings [SHL] 103
6.64.1 Description of application vulnerability 103
6.64.2 Cross reference 103
6.64.3 Mechanism of failure 103
6.64.4 Applicable language characteristics 104
6.64.5 Avoiding the vulnerability or mitigating its effects 104
6.64.6 Implications for language design and evolution 104

7 Application vulnerabilities 105
7.1 General 105
7.2 Unrestricted file upload [CBF] 105
7.2.1 Description of application vulnerability 105
7.2.2 Cross reference 105
7.2.3 Mechanism of failure 105
7.2.4 Avoiding the vulnerability or mitigating its effects 105
7.3 Download of code without integrity check [DLB] 106
7.3.1 Description of application vulnerability 106
7.3.2 Cross reference 106
7.3.3 Mechanism of failure 106
7.3.4 Avoiding the vulnerability or mitigating its effects 106
7.4 Executing or loading untrusted code [XYS] 107
7.4.1 Description of application vulnerability 107
7.4.2 Cross reference 107
7.4.3 Mechanism of failure 107
7.4.4 Avoiding the vulnerability or mitigating its effects 107
7.5 Inclusion of functionality from untrusted control sphere [DHU] 108
7.5.1 Description of application vulnerability 108
7.5.2 Cross reference 108
7.5.3 Mechanism of failure 108
7.5.4 Avoiding the vulnerability or mitigating its effects 108
7.6 Use of unchecked data from an uncontrolled or tainted source [EFS] 108
7.6.1 Description of application vulnerability 108
7.6.2 Cross reference 109
7.6.3 Mechanism of failure 109
7.6.4 Avoiding the vulnerability or mitigating its effects 109
7.7 Cross-site scripting [XYT] 109
7.7.1 Description of application vulnerability 109
7.7.2 Cross reference 109
7.7.3 Mechanism of failure 110
7.7.4 Avoiding the vulnerability or mitigating its effects 111
7.8 URL redirection to untrusted site ("open redirect") [PYQ] 112
7.8.1 Description of application vulnerability 112
7.8.2 Cross reference 112
7.8.3 Mechanism of failure 112
7.8.4 Avoiding the vulnerability or mitigating its effects 112
7.9 Injection [RST] 112
7.9.1 Description of application vulnerability 112
7.9.2 Cross reference 113
7.9.3 Mechanism of failure 114
7.9.4 Avoiding the vulnerability or mitigating its effects 115
7.10 Unquoted search path or element [XZQ] 115
7.10.1 Description of application vulnerability 115
7.10.2 Cross reference 115
7.10.3 Mechanism of failure 115
7.10.4 Avoiding the vulnerability or mitigating its effects 115
7.11 Path traversal [EWR] 116
7.11.1 Description of application vulnerability 116

© ISO/IEC 2019 – All rights reserved

7.11.2 Cross reference ... 116
7.11.3 Mechanism of failure ... 116
7.11.4 Avoiding the vulnerability or mitigating its effects ... 118
7.12 Resource names [HTS] ... 118
7.12.1 7Description of application vulnerability ... 118
7.12.2 Cross reference ... 119
7.12.3 Mechanism of Failure ... 119
7.12.4 Avoiding the vulnerability or mitigating its effects ... 119
7.13 Resource exhaustion [XZP] ... 119
7.13.1 Description of application vulnerability ... 119
7.13.2 Cross reference ... 119
7.13.3 Mechanism of failure ... 119
7.13.4 Avoiding the vulnerability or mitigating its effects ... 120
7.14 Authentication logic error [XZO] ... 120
7.14.1 Description of application vulnerability ... 120
7.14.2 Cross reference ... 120
7.14.3 Mechanism of failure ... 121
7.14.4 Avoiding the vulnerability or mitigating its effects ... 122
7.15 Improper restriction of excessive authentication attempts [WPL] ... 122
7.15.1 Description of application vulnerability ... 122
7.15.2 Cross reference ... 122
7.15.3 Mechanism of failure ... 122
7.15.4 Avoiding the vulnerability or mitigating its effects ... 122
7.16 Hard-coded credentials [XYP] ... 123
7.16.1 Description of application vulnerability ... 123
7.16.2 Cross reference ... 123
7.16.3 Mechanism of failure ... 123
7.16.4 Avoiding the vulnerability or mitigating its effects ... 123
7.17 Insufficiently protected credentials [XYM] ... 124
7.17.1 Description of application vulnerability ... 124
7.17.2 Cross reference ... 124
7.17.3 Mechanism of failure ... 124
7.17.4 Avoiding the vulnerability or mitigating its effects ... 124
7.18 Missing or inconsistent access control [XZN] ... 125
7.18.1 Description of application vulnerability ... 125
7.18.2 Cross reference ... 125
7.18.3 Mechanism of failure ... 125
7.18.4 Avoiding the vulnerability or mitigating its effects ... 125
7.19 Incorrect authorization [BJE] ... 125
7.19.1 Description of application vulnerability ... 125
7.19.2 Cross reference ... 125
7.19.3 Mechanism of failure ... 125
7.19.4 Avoiding the vulnerability or mitigating its effects ... 126
7.20 Adherence to least privilege [XYN] ... 126
7.20.1 Description of application vulnerability ... 126
7.20.2 Cross reference ... 126
7.20.3 Mechanism of failure ... 126
7.20.4 Avoiding the vulnerability or mitigating its effects ... 126
7.21 Privilege sandbox issues [XYO] ... 127
7.21.1 Description of application vulnerability ... 127
7.21.2 Cross reference ... 127
7.21.3 Mechanism of failure ... 127
7.21.4 Avoiding the vulnerability or mitigating its effects ... 128
7.22 Missing required cryptographic step [XZS] ... 128
7.22.1 Description of application vulnerability ... 128
7.22.2 Cross reference ... 128
7.22.3 Mechanism of failure ... 128
7.22.4 Avoiding the vulnerability or mitigating its effects ... 128

 © ISO/IEC 2019 – All rights reserved

7.23 Improperly verified signature [XZR] 129
7.23.1 Description of application vulnerability 129
7.23.2 Cross reference 129
7.23.3 Mechanism of failure 129
7.23.4 Avoiding the vulnerability or mitigating its effects 129
7.24 Use of a one-way hash without a salt [MVX] 129
7.24.1 Description of application vulnerability 129
7.24.2 Cross reference 129
7.24.3 Mechanism of failure 129
7.24.4 Avoiding the vulnerability or mitigating its effects 130
7.25 Inadequately secure communication of shared resources [CGY] 130
7.25.1 Description of application vulnerability 130
7.25.2 Cross-references 130
7.25.3 Mechanism of failure 130
7.25.4 Avoiding the vulnerability or mitigating its effect 131
7.26 Memory locking [XZX] 131
7.26.1 Description of application vulnerability 131
7.26.2 Cross reference 131
7.26.3 Mechanism of failure 132
7.26.4 Avoiding the vulnerability or mitigating its effects 132
7.27 Sensitive information not cleared before use [XZK] 132
7.27.1 Description of application vulnerability 132
7.27.2 Cross reference 132
7.27.3 Mechanism of failure 132
7.27.4 Avoiding the vulnerability or mitigating its effects 133
7.28 Time consumption measurement [CCM] 133
7.28.1 Description of application vulnerability 133
7.28.2 Cross-references 133
7.28.3 Mechanism of failure 133
7.28.4 Avoiding the vulnerability or mitigating its effect 133
7.29 Discrepancy information leak [XZL] 134
7.29.1 Description of application vulnerability 134
7.29.2 Cross reference 134
7.29.3 Mechanism of failure 134
7.29.4 Avoiding the vulnerability or mitigating its effects 134
7.30 Unspecified functionality [BVQ] 135
7.30.1 Description of application vulnerability 135
7.30.2 Cross reference 135
7.30.3 Mechanism of failure 135
7.30.4 Avoiding the vulnerability or mitigating its effects 135
7.31 Fault tolerance and failure strategies [REU] 136
7.31.1 Description of application vulnerability 136
7.31.2 Cross reference 136
7.31.3 Mechanism of failure 137
7.31.4 Avoiding the vulnerability or mitigating its effects 137
7.32 Distinguished values in data types [KLK] 138
7.32.1 Description of application vulnerability 138
7.32.2 Cross reference 138
7.32.3 Mechanism of failure 138
7.32.4 Avoiding the vulnerability or mitigating its effects 139
7.33 Clock issues [CCI] 139
7.33.1 Description of application vulnerability 139
7.33.2 Cross-references 140
7.33.3 Mechanism of failure 140
7.33.4 Avoiding the vulnerability or mitigating its effect 141
7.34 Time drift and jitter [CDJ] 141
7.34.1 Description of application vulnerability 141
7.34.2 Cross-references 142

© ISO/IEC 2019 – All rights reserved

7.34.3 Mechanism of failure 142
7.34.4 Avoiding the vulnerability or mitigating its effect 142

8 New vulnerabilities 143
8.1 General 143
8.2 Modifying constants [UJO] 143
8.2.1 Description of application vulnerability 143
8.2.2 Cross reference 143
8.2.3 Mechanism of failure 143
8.2.4 Applicable language characteristics 144
8.2.5 Avoiding the vulnerability or mitigating its effects 144
8.2.6 Implications for language design and evolution 144

Annex A (informative) **Vulnerability taxonomy and list 145**

Annex B (informative) **Selected guidance to language designers 153**

Annex C (informative) **Language-specific vulnerability template 155**

Bibliography 158

Index 161

 © ISO/IEC 2019 – All rights reserved

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document can be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html.

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

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

This first edition cancels and replaces ISO IEC TR 24772:2013, which has been split into several parts.

A list of all parts in the ISO/IEC 24772 series can be found on the ISO website.

© ISO/IEC 2019 – All rights reserved

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 can 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 endanger the safety of a system or be exploited by attackers to compromise the security or privacy of a system.

This document provides users of programming languages with a language-independent overview of potential vulnerabilities in their usage and ways to avoid or mitigate them. Other parts of the ISO/IEC 24772 series describe how the general observations apply to specific languages.

This document 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.

The intended audience for this document are those who are concerned with assuring the predictable execution of the software of their system; that is, those who are developing, qualifying, or maintaining a software system and need to avoid language constructs that can cause the software to execute in a manner other than intended.

Developers of applications that have clear safety, security or mission-criticality requirements are expected to be aware of the risks associated with their code and can use this document to ensure that their development practices address the issues presented by the chosen programming languages, for example by subsetting or providing coding guidelines.

Specific audiences for this document include in particular developers, maintainers and regulators of:

— safety-critical applications that can cause loss of life, human injury, or damage to the environment;

— security-critical applications that need to ensure properties of confidentiality, integrity, and availability;

— mission-critical applications that need to avoid loss or damage to property or finance;

— business-critical applications where correct operation is essential to the successful operation of the business;

— scientific, modelling and simulation applications that require high confidence in the results of possibly complex, expensive and extended calculation.

However, it should not be assumed that other developers can ignore this document. A weakness in a non-critical application can provide the route by which an attacker gains control of a system or otherwise disrupts co-hosted applications that are critical. It is hoped that all developers would use this document to ensure that common vulnerabilities are removed or at least minimized from all applications.

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

It should be noted that this document 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.

 © ISO/IEC 2019 – All rights reserved

Programming languages — Guidance to avoiding vulnerabilities in programming languages —

Part 1: Language-independent guidance

1 Scope

This document 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. Language-specific descriptions of these vulnerabilities are provided in other parts of the ISO/IEC 24772 series.

It is applicable to the software developed, reviewed, or maintained for any application.

This document 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 and applications to be assured are not treated.

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

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 2382, *Information technology — Vocabulary*

ISO 80000-2, *Quantities and units — Part 2: Mathematics*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 2382, ISO 80000-2 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1 Terms related to communication

3.1.1
protocol
set of rules and supporting structures for the interaction of threads

Note 1 to entry: A protocol can be tightly embedded and rely on data in memory and hardware to control interaction of threads or can be applied to more loosely coupled arrangements, such as message communication spanning networks and computer systems.

© ISO/IEC 2019 – All rights reserved

3.1.2
stateless protocol
communication or cooperation between threads where no state is preserved in the protocol itself (example HTTP or direct access to a shared resource)

Note 1 to entry: Since most interaction between threads requires that state be preserved, the cooperating threads needs to use values of the resources(s) themselves or add additional communication exchanges to maintain state. Stateless protocols require that the application provide explicit resource protection and locking mechanisms to guarantee the correct creation, view, access to, modification of, and destruction of the resource – for example, the state needed for correct handling of the resource.

3.2 Terms related to execution model

3.2.1
thread
sequential stream of execution

Note 1 to entry: Although the term thread is used here and the context portrayed is that of shared-memory threads executing as part of a process, everything documented applies equally to other variants of concurrency such as interrupt handlers being enabled by a process, processes being created on the same system using operating system routines, or processes created as a result of distributed messages sent over a network. The mitigation approaches will be similar to those listed in the relevant vulnerability descriptions, but the implications for standardization would be dependent on how much language support is provided for the programming of the concurrent system.

3.2.2
thread activation
creation and setup of a thread up to the point where the thread begins execution

Note 1 to entry: A thread can depend on one or more other threads to define its access to other objects to be accessed and to determine its duration.

3.2.3
activated thread
thread that is created and then begins execution as a result of thread activation

3.2.4
activating thread
thread that exists first and makes the library calls or contains the language syntax that causes the activated thread to be activated

Note 1 to entry: The activating thread may wait, or not, for the activated thread to finish activation and may or may not check for errors if the activation fails. The activating thread can be permitted, or not, to terminate until after the activated thread terminates.

3.2.5
static thread activation
creation and initiation of a thread by program initiation, an operating system or runtime kernel, or by another thread as part of a declarative part of the thread before it begins execution

Note 1 to entry: In static activation, a static analysis can determine exactly how many threads will be created and how much resource, in terms of memory, processors, CPU cycles, priority ranges and inter-thread communication structures, will be needed by the executing program before the program begins.

3.2.6
dynamic thread activation
creation and initiation of a thread by another thread (including the main program) as an executable, repeatable command, statement or subprogram call

© ISO/IEC 2019 – All rights reserved

3.2.7
thread abort
request to stop and shut down a thread immediately

Note 1 to entry: The request is asynchronous if from another thread, or synchronous if from the thread itself. The effect of the abort request (such as whether it is treated as an exception) and its immediacy (that is, how long the thread can continue to execute before it is shut down) depend on language-specific rules. Immediate shutdown minimizes latency but can leave shared data structures in a corrupted state.

3.2.8
termination-directing thread
thread (including the OS) that requests the abortion of one or more threads

3.2.9
thread termination
completion and orderly shutdown of a thread, where the thread is permitted to make data objects consistent, release any acquired resources, and notify any dependent threads that it is terminating

Note 1 to entry: There are a number of steps in the termination of a thread as listed below, but depending on the multithreading model, some of these steps can be combined, explicitly programmed, or missing:

— the termination of programmed execution of the thread, including termination of any synchronous communication;

— the finalization of the local objects of the thread;

— waiting for any threads that can depend on the thread to terminate;

— finalization of any state associated with dependent threads;

— notification that finalization is complete, including possible notification of the activating task;

— removal and clean-up of thread control blocks and any state accessible by the thread or by other threads in outer scopes.

3.2.10
terminated thread
thread that has been halted from any further execution

3.2.11
master thread
thread which needs to wait for a terminated thread before it can take further execution steps (including termination of itself)

3.2.12
process
single execution of a program, or portion of an application

Note 1 to entry: Processes do not normally share a common memory space, but often share:

— processor;

— network;

— operating system;

— filing system;

— environment variables; or

— other resources.

Processes are usually started and stopped by an operating system and may or may not interact with other processes. A process can contain multiple threads.

© ISO/IEC 2019 – All rights reserved

3.3 Properties

3.3.1
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

3.3.2
predictable execution
property of the program such that all possible executions have results that can be predicted from the source code

3.4 Safety

3.4.1
safety hazard
potential source of harm

Note 1 to entry: IEC 61508–4[20] 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*. 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*).

3.4.2
safety-critical software
software for applications where failure can cause very serious consequences such as human injury or death

Note 1 to entry: IEC 61508–3[20] defines *Safety-related software* as *software that is used to implement safety functions in a safety-related system*. Although, in some domains a distinction is made between safety-related (can lead to any harm) and safety-critical (life threatening), this document uses the term *safety-critical* for all vulnerabilities that can result in safety hazards.

3.5 Vulnerabilities

3.5.1
application vulnerability
security vulnerability or safety hazard, or defect

3.5.2
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 to entry: 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.

3.5.3
security vulnerability
weakness in an information system, system security procedures, internal controls, or implementation that can be exploited or triggered by a threat

 © ISO/IEC 2019 – All rights reserved

3.5.4
failure
malfunction of the system or component which has as subcategories omission failure, commission failure, timing failure and value failure

3.5.5
omission failure
service that is requested but never rendered

Note 1 to entry: The client can wait forever or can be notified about the failure (termination) of the service.

3.5.6
commission failure
service that initiates unexpected actions, e. g., communication that is unexpected by the receiver

Note 1 to entry: The service can wait forever, causing omission failures for subsequent calls by clients. The receiver can be hindered to do its legitimate actions in time. At a minimum, resources are consumed that are possibly needed by others.

3.5.7
timing failure
service that is not rendered before an imposed deadline

Note 1 to entry: This results in a system response that is (too) late, causing corresponding damages to the real world affected by the system.

3.1.5.8
value failure
service delivers incorrect or tainted results

Note 1 to entry: The client continues computations with these corrupted values, causing a spread of consequential application errors.

4 Applying this document

This document 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 can 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 can want to consider the vulnerabilities inherent in various candidate languages;

— scientists, engineers, economists, statisticians, or others who write computer programs as tools of their chosen craft can read this document to become more familiar with the issues that can affect their work.

There are a number of ways to avoid a vulnerability:

— avoiding the particular coding constructs that are found to be problematic;

— using static analysis tools to detect anomalous situations, including usage of a compiler that provides warnings if a construct is problematic;

— choosing a programming language that avoids or mitigates a class of vulnerabilities;

© ISO/IEC 2019 – All rights reserved