

ISO/IEC JTC 1/SC 22/WG 23 N 0275

Draft language-specific annex for SPARK

Date 2 September 2010
Contributed by SC 22/WG 9
Original file name N502, SPARK_Annex Draft 1, 16 Aug 2010.doc
Notes

1 Annex SPARK – Final 2 Draft

3 SPARK.Specific 4 information for 5 vulnerabilities

6 7 Status and History

8 *September 2009: First draft from SPARK*
9 *team.*

10 *November 2009: Second draft following*
11 *comments from HRG.*

12 *May 2010: Updates to be consistent with*
13 *Ada Annex and new vulnerabilities in the*
14 *parent TR.*

15 *June 2010: Updates following review*
16 *comments from HRG.*

17 *July 2010: Submit to WG9.*

18 SPARK.1 Identification of 19 standards and associated 20 documentation

21 See Ada.1, plus the references below. In the
22 body of this annex, the following documents
23 are referenced using the short abbreviation
24 that introduces each document, optionally
25 followed by a specific section number. For
26 example “[SLRM 5.2]” refers to section 5.2
27 of the SPARK Language Definition.

28
29 [SLRM] [SPARK Language Definition:](#)
30 “SPARK95: The SPADE Ada Kernel
31 (Including RavenSPARK)” Latest version
32 always available from [www.altran-](http://www.altran-praxis.com)
33 [praxis.com](http://www.altran-praxis.com).

34
35 [SB] “High Integrity Software: The SPARK
36 Approach to Safety and Security.” John
37 Barnes. Addison-Wesley, 2003. ISBN 0-321-
38 13616-0.

39
40 [IFA] “Information-Flow and Data-Flow
41 Analysis of while-Programs.” Bernard Carré
42 and Jean-Francois Bergeretti, ACM
43 Transactions on Programming Languages
44 and Systems (TOPLAS) Vol. 7 No. 1,
45 January 1985. pp 37-61.

46
47 [LSP] “A behavioral notion of subtyping.”
48 Barbara Liskov and Jeannette Wing. ACM

49 Transactions on Programming Languages
50 and Systems (TOPLAS), Volume 16, Issue 6
51 (November 1994), pp. 1811 - 1841.
52

53 SPARK.2 General terminology 54 and concepts

55 The SPARK language is a contractualized
56 subset of Ada, specifically designed for high-
57 assurance systems. SPARK is designed to
58 be amenable to various forms of static
59 analysis that prevent or mitigate the
60 vulnerabilities described in this TR.

61 This section introduces concepts and
62 terminology which are specific to SPARK
63 and/or relate to the use of static analysis
64 tools.
65

66 67 Soundness

68 This concept relates to the absence of false-
69 negative results from a static analysis tool. A
70 false negative is when a tool is posed the
71 question “Does this program exhibit
72 vulnerability X?” but incorrectly responds
73 “no.” Such a tool is said to be **unsound** for
74 vulnerability X. A sound tool effectively finds
75 **all** the vulnerabilities of a particular class,
76 whereas an unsound tool only finds some of
77 them.
78

79 The provision of soundness in static analysis
80 is problematic, mainly owing to the presence
81 of unspecified and undefined features in
82 programming languages. Claims of
83 soundness made by tool vendors should be
84 carefully evaluated to verify that they are
85 reasonable for a particular language,
86 compilers and target machines. Soundness
87 claims are always underpinned by
88 assumptions (for example, regarding the
89 reliability of memory, the correctness of
90 compiled code and so on) that should also
91 be validated by users for their
92 appropriateness.

93
94 Static analysis techniques can also be
95 **sound in theory** – where the mathematical
96 model for the language semantics and
97 analysis techniques have been formally
98 stated, proved, and reviewed – but
99 **unsound in practice** owing to defects in the
100 implementation of analysis tools. Again,
101 users should seek evidence to support any
102 soundness claim made by language

1 designers and tool vendors. A language
 2 which is **unsound in theory** can never be
 3 sound in practice.
 4
 5 The single overriding design goal of SPARK
 6 is the provision of a static analysis
 7 framework which is **sound in theory**, and
 8 as **sound in practice** as is reasonably
 9 possible.
 10
 11 In the subsections below, we say that
 12 SPARK **prevents** a vulnerability if supported
 13 by a form of static analysis which is sound in
 14 theory. Otherwise, we say that SPARK
 15 **mitigates** a particular vulnerability.
 16
 17 **SPARK Processor**
 18 We define a “SPARK Processor” to be a tool
 19 that implements the various forms of static
 20 analysis required by the SPARK language
 21 definition. Without a SPARK Processor, a
 22 program cannot reasonably be claimed to be
 23 SPARK at all, much in the same way as a
 24 compiler checks the static semantic rules of
 25 a standard programming language.
 26
 27 In SPARK, certain forms of analysis are said
 28 to be **mandatory** – they are required to be
 29 implemented and programs must pass these
 30 checks to be valid SPARK. Examples of
 31 mandatory analyses are the enforcement of
 32 the SPARK language subset, static
 33 semantic analysis (e.g. enhanced type
 34 checking) and information flow analysis
 35 [IFA].
 36
 37 Some analyses are said to be **optional** – a
 38 user may choose to enable these additional
 39 analyses at their discretion. The most
 40 notable example of an optional analysis in
 41 SPARK is the generation of verification
 42 conditions and their proof using a theorem
 43 proving tool. Optional analyses may provide
 44 greater depth of analysis, protection from
 45 additional vulnerabilities, and so on, at the
 46 cost of greater analysis time and effort.
 47
 48 **Failure modes for static analysis**
 49 Unlike a language compiler, a user can
 50 always choose not to, or might just forget to
 51 run a static analysis tool. Therefore, there
 52 are two modes of failure that apply to all
 53 vulnerabilities:
 54

- 55
 56
 57
 58
 59
 60
 61
1. The user fails to apply the appropriate static analysis tool to their code.
 2. The user fails to review or misinterprets the output of static analysis.

62 **SPARK.3.BRS Obscure** 63 **Language Features [BRS]**

64 SPARK mitigates this vulnerability.

65 **SPARK.3.BRS.1 Terminology** 66 **and features**

67 As in Ada.3.BRS.1.

68 **SPARK.3.BRS.2 Description of** 69 **vulnerability**

70 As in Ada.3.BRS.2.

71 **SPARK.3.BRS.3 Avoiding the** 72 **vulnerability or mitigating its** 73 **effects**

74 The design of the SPARK subset avoids
 75 many language features that might be said
 76 to be “obscure” or “hard to understand”,
 77 such as controlled types, unrestricted
 78 tasking, anonymous access types and so
 79 on.

80
 81 SPARK goes further, though, in aiming for a
 82 completely *unambiguous* semantics,
 83 removing all erroneous and implementation-
 84 dependent features from the language. This
 85 means that a SPARK program should have
 86 a single meaning to programmers,
 87 reviewers, maintainers and all compilers.

88
 89 SPARK also bans the aliasing, overloading,
 90 and redeclaration of names, so that one
 91 entity only ever has one name and one
 92 name can denote at most one entity, further
 93 reducing the risk of mis-understanding or
 94 mis-interpretation of a program by a person,
 95 compiler or other tools.

96 **SPARK.3.BRS.4 Implications** 97 **for standardization**

98 None.

1 **SPARK.3.BRS.5 Bibliography**

2 None.

3 **SPARK.3.BQF Unspecified
4 Behaviour [BQF]**

5 SPARK prevents this vulnerability.

6 **SPARK.3.BQF.1 Terminology
7 and features**

8 As in Ada.3.BQF.1.

9 **SPARK.3.BQF.2 Description of
10 vulnerability**

11 As in Ada.3.BQF.2.

12 **SPARK.3.BQF.3 Avoiding the
13 vulnerability or mitigating its
14 effects**

15 SPARK is designed to eliminate all
16 unspecified language features and bounded
17 errors, either by subsetting to make the
18 offending language feature illegal in SPARK,
19 or by ensuring that the language has neutral
20 semantics with regard to an unspecified
21 behaviour.

22
23 “Neutral semantics” means that the program
24 has identical meaning regardless of the
25 choice made by a compiler for a particular
26 unspecified language feature.

27
28 For example:

- 29 • Unspecified behaviour as a result of
30 parameter-passing mechanism is
31 avoided through subsetting (no
32 access types) and analysis to make
33 sure that formal and global
34 parameters do not overlap and
35 create a potential for aliasing [SLRM
36 6.4].
 - 37
38 • Dependence on evaluation order is
39 prevented through analysis so that
40 all expressions in SPARK are free of
41 side-effects and potential run-time
42 errors. Therefore, any evaluation
43 order is allowed and the result of the
44 evaluation is the same in all cases
45 [SLRM 6.1].
- 46

47

48

49

50

51

- Bounded error as a result of uninitialized variables is prevented by application of static information flow analysis [IFA].

52 **SPARK.3.BQF.4 Implications
53 for standardization**

54 None.

55 **SPARK.3.BQF.5 Bibliography**

56 None.

57 **SPARK.3.EWF Undefined
58 Behaviour [EWF]**

59 SPARK prevents this vulnerability.

60 **SPARK.3.EWF.1 Terminology
61 and features**

62 As in Ada.3.EWF.1.

63 **SPARK.3.EWF.2 Description of
64 vulnerability**

65 As in Ada.3.EWF.2.

66 **SPARK.3.EWF.3 Avoiding the
67 vulnerability or mitigating its
68 effects**

69 SPARK prevents all erroneous behaviour,
70 either through subsetting or static analysis
71 [SB 1.3].

72 **SPARK.3.EWF.4 Implications
73 for standardization**

74 None.

75 **SPARK.3.EWF.5 Bibliography**

76 None.

77 **SPARK.3.FAB Implementation-
78 Defined Behaviour [FAB]**

79 SPARK mitigates this vulnerability.

1 **SPARK.3.FAB.1 Terminology**
2 **and features**

3 As in Ada.3.FAB.1.

4 **SPARK.3.FAB.2 Description of**
5 **vulnerability**

6 As in Ada.3.FAB.2.

7 **SPARK.3.FAB.3 Avoiding the**
8 **vulnerability or mitigating its**
9 **effects**

10 SPARK allows a number of implementation-
11 defined features as in Ada. These include:

12

- 13 • The range of predefined integer
14 types.
- 15 • The range and precision of
16 predefined floating-point types.
- 17 • The range of System.Any_Priority
18 and its subtypes.
- 19 • The value of constants such as
20 System.Max_Int, System.Min_Int
21 and so on.
- 22 • The selection of T'Base for a user-
23 defined integer or floating-point type
24 T.
- 25 • The rounding mode of floating-point
26 types.

27

28 In the first four cases, static analysis tools
29 can be configured to “know” the appropriate
30 values [SB 9.6]. Care must be taken to
31 ensure that these values are correct for the
32 intended implementation. In the fifth case,
33 SPARK defines a contract to indicate the
34 choice of base-type, which can be checked
35 by a pragma Assert. In the final case,
36 additional static analysis of numerical
37 precision must be performed by the user to
38 ensure the correctness of floating-point
39 algorithms.

40 **SPARK.3.FAB.4 Implications for**
41 **standardization**

42 None.

43 **SPARK.3.FAB.5 Bibliography**

44 None.

45 **SPARK.3.MEM Deprecated**
46 **Language Features [MEM]**

47 SPARK is identical to Ada with respect to
48 this vulnerability and its mitigation. See
49 Ada.3.MEM.

50 **SPARK.3.NMP Pre-Processor**
51 **Directives [NMP]**

52 SPARK is identical to Ada with respect to
53 this vulnerability and its mitigation. See
54 Ada.3.NMP.

55 **SPARK.3.NAI Choice of Clear**
56 **Names [NAI]**

57 SPARK is identical to Ada with respect to
58 this vulnerability and its mitigation. See
59 Ada.3.NAI.

60 **SPARK.3.AJN Choice of**
61 **Filenames and other External**
62 **Identifiers [AJN]**

63 SPARK is identical to Ada with respect to
64 this vulnerability and its mitigation. See
65 Ada.3.AJN.

66 **SPARK.3.XYR Unused Variable**
67 **[XYR]**

68 SPARK mitigates this vulnerability.

69 **SPARK.3.XYR.1 Terminology**
70 **and features**

71 As in Ada.3.XYR.1.

72 **SPARK.3.XYR.2 Description of**
73 **vulnerability**

74 As in Ada.3.XYR.2.

75 **SPARK.3.XYR.3 Avoiding the**
76 **vulnerability or mitigating its**
77 **effects**

78 As in Ada.3.XYR.3. Also, SPARK is
79 designed to permit sound static analysis of
80 the following cases [IFA]:

81

- 82 • Variables which are declared but not
83 used at all.

- 1 • Variables which are assigned to, but
2 the resulting value is not used in any
3 way that affects an output of the
4 enclosing subprogram. This is called
5 an “ineffective assignment” in
6 SPARK.

7 **SPARK.3.XYR.4 Implications for**
8 **standardization**

9 None.

10 **SPARK.3.XYR.5 Bibliography**

11 None.

12 **SPARK.3.YOW Identifier Name**
13 **Reuse [YOW]**

14 SPARK prevents this vulnerability.

15 **SPARK.3.YOW.1 Terminology**
16 **and features**

17 As in Ada.3.YOW.1.

18 **SPARK.3.YOW.2 Description of**
19 **vulnerability**

20 As in Ada.3.YOW.2.

21 **SPARK.3.YOW.3 Avoiding the**
22 **vulnerability or mitigating its**
23 **effects**

24 This vulnerability is prevented through
25 language rules enforced by static analysis.
26 SPARK does not permit names in local
27 scopes to redeclare and hide names that are
28 already visible in outer scopes [SLRM 6.1].

29 **SPARK.3.YOW.4 Implications**
30 **for standardization**

31 None.

32 **SPARK.3.YOW.5 Bibliography**

33 None.

34 **SPARK.3.BKL Namespace**
35 **Issues [BJL]**

36 SPARK is identical to Ada with respect to
37 this vulnerability and its mitigation. See
38 Ada.3.BJL.

39 **SPARK.3.IHN Type System**
40 **[IHN]**

41 SPARK mitigates this vulnerability.

42 **SPARK.3.IHN.1 Terminology**
43 **and features**

44 SPARK’s type system is a simplification of
45 that of Ada. Both Explicit and Implicit
46 conversions are permitted in SPARK, as is
47 instantiation and use of
48 `Unchecked_Conversion` [SB 1.3].
49

50 A design goal of SPARK is the provision of
51 *static type safety*, meaning that programs
52 can be shown to be free from all run-time
53 type failures using entirely static analysis. If
54 this optional analysis is achieved, a SPARK
55 program should never raise an exception at
56 run-time.

57 **SPARK.3.IHN.2 Description of**
58 **vulnerability**

59 As in Ada.3.IHN.2 for
60 `Unchecked_Conversion`.

61 **SPARK.3.IHN.3 Avoiding the**
62 **vulnerability or mitigating its**
63 **effects**

64 Vulnerabilities relating to value conversions,
65 exceptions, and assignments are mitigated
66 by static analysis. Vulnerabilities relating to
67 the use of `Unchecked_Conversion` are as in
68 Ada.

69 **SPARK.3.IHN.4 Implications for**
70 **standardization**

71 None.

72 **SPARK.3.IHN.5 Bibliography**

73 None.

1 **SPARK.3.STR Bit**
2 **Representation [STR]**

3 SPARK mitigates this vulnerability.

4 **SPARK.3.STR.1 Terminology**
5 **and features**

6 As in Ada.3.STR.1.

7 **SPARK.3.STR.2 Description of**
8 **vulnerability**

9 SPARK is designed to offer a semantics
10 which is independent of the underlying
11 representation chosen by a compiler for a
12 particular target machine. Representation
13 clauses are permitted, but these do not
14 affect the semantics as seen by a static
15 analysis tool [SB 1.3].

16 **SPARK.3.STR.3 Avoiding the**
17 **vulnerability or mitigating its**
18 **effects**

19 As in Ada.3.STR.4.

20 **SPARK.3.STR.4 Implications for**
21 **standardization**

22 None.

23 **SPARK.3.STR.5 Bibliography**

24 None.

25 **SPARK.3.PLF Floating-point**
26 **Arithmetic [PLF]**

27 SPARK is identical to Ada with respect to
28 this vulnerability and its mitigation. See
29 Ada.3.PLF.

30 **SPARK.3.CCB Enumerator**
31 **Issues [CCB]**

32 SPARK is identical to Ada with respect to
33 this vulnerability and its mitigation. See
34 Ada.3.CCB.

35 **SPARK.3.FLC Numeric**
36 **Conversion Errors [FLC]**

37 SPARK prevents this vulnerability.

38 **SPARK.3.FLC.1 Terminology**
39 **and features**

40 As in Ada.3.FLC.1.

41 **SPARK.3.FLC.2 Description of**
42 **vulnerability**

43 As in Ada.3.FLC.2.

44 **SPARK.3.FLC.3 Avoiding the**
45 **vulnerability or mitigating its**
46 **effects**

47 SPARK is designed to be amenable to static
48 verification of the absence of predefined
49 exceptions, and in particular all cases
50 covered by this vulnerability [SB 11]. All
51 numeric conversions (both explicit and
52 implicit) give rise to a verification condition
53 that must be discharged, typically using an
54 automated theorem-prover.

55 **SPARK.3.FLC.4 Implications for**
56 **standardization**

57 None.

58 **SPARK.3.FLC.5 Bibliography**

59 None.

60 **SPARK.3.CJM String**
61 **Termination [CJM]**

62 SPARK is identical to Ada with respect to
63 this vulnerability and its mitigation. See
64 Ada.3.CJM.

65 **SPARK.3.XYX Boundary**
66 **Beginning Violation [XYX]**

67 SPARK prevents this vulnerability.

68 **SPARK.3.XYX.1 Terminology**
69 **and features**

70 As in Ada.3.XYX.1.

71 **SPARK.3.XYX.2 Description of**
72 **vulnerability**

73 As in Ada.3.XYX.2.

1 **SPARK.3.XYX.3 Avoiding the**
 2 **vulnerability or mitigating its**
 3 **effects**

4 SPARK is designed to permit static analysis
 5 for all such boundary violations, through
 6 techniques such as theorem proving or
 7 abstract interpretation [SB 11].

8
 9 SPARK programs that have been subject to
 10 this level of analysis can be compiled with
 11 run-time checks suppressed, supported by a
 12 body of evidence that such checks could
 13 never fail, and thus removing the possibility
 14 of erroneous execution.

15 **SPARK.3.XYX.4 Implications for**
 16 **standardization**

17 None.

18 **SPARK.3.XYX.5 Bibliography**

19 None.

20 **SPARK.3.XYZ Unchecked Array**
 21 **Indexing [XYZ]**

22 SPARK prevents this vulnerability.

23 **SPARK.3.XYZ.1 Terminology**
 24 **and features**

25 As in Ada.3.XYZ.1.

26 **SPARK.3.XYZ.2 Description of**
 27 **vulnerability**

28 As in Ada.3.XYZ.2.

29 **SPARK.3.XYZ.3 Avoiding the**
 30 **vulnerability or mitigating its**
 31 **effects**

32 As per SPARK.3.XYX.3 – this vulnerability is
 33 eliminated in SPARK by static analysis using
 34 the same techniques.

35 **SPARK.3.XYZ.4 Implications for**
 36 **standardization**

37 None.

38 **SPARK.3.XYZ.5 Bibliography**

39 None.

40 **SPARK.3.XYW Unchecked**
 41 **Array Copying [XYW]**

42 SPARK prevents this vulnerability.

43 **SPARK.3.XYW.1 Terminology**
 44 **and features**

45 As in Ada.3.XYW.1.

46 **SPARK.3.XYW.2 Description of**
 47 **vulnerability**

48 As in Ada.3.XYW.2.

49 **SPARK.3.XYW.3 Avoiding the**
 50 **vulnerability or mitigating its**
 51 **effects**

52 Array assignments in SPARK are only
 53 permitted between objects that have
 54 statically matching bounds, so there is no
 55 possibility of an exception being raised [SB
 56 5.5, SLRM 4.1.2]. Ada's "slicing" and
 57 "sliding" of arrays is not permitted in SPARK,
 58 so this vulnerability cannot occur.

59 **SPARK.3.XYW.4 Implications**
 60 **for standardization**

61 None.

62 **SPARK.3.XYW.5 Bibliography**

63 None.

64 **SPARK.3.XZB Buffer Overflow**
 65 **[XZB]**

66 SPARK prevents this vulnerability.

67 **SPARK.3.XZB.1 Terminology**
 68 **and features**

69 As in Ada.3.HCF.1.

70 **SPARK.3.XZB.2 Description of**
 71 **vulnerability**

72 As in Ada.3.XZB.2.

1 **SPARK.3.XZB.3 Avoiding the**
 2 **vulnerability or mitigating its**
 3 **effects**

4 As per SPARK.3.XYX.3 – this vulnerability is
 5 eliminated in SPARK by static analysis using
 6 the same techniques.

7 **SPARK.3.XZB.4 Implications for**
 8 **standardization**

9 None.

10 **SPARK.3.XZB.5 Bibliography**

11 None.

12 **SPARK.3.HCF Pointer Casting**
 13 **and Pointer Type Changes**
 14 **[HCF]**

15 SPARK prevents this vulnerability.

16 **SPARK.3.HCF.1 Terminology**
 17 **and features**

18 As in Ada.3.HCF.1.

19 **SPARK.3.HCF.2 Description of**
 20 **vulnerability**

21 As in Ada.3.HCF.2.

22 **SPARK.3.HCF.3 Avoiding the**
 23 **vulnerability or mitigating its**
 24 **effects**

25 This vulnerability cannot occur in SPARK,
 26 since the SPARK subset forbids the
 27 declaration or use of access (pointer) types
 28 [SB 1.3, SLRM 3.10].

29 **SPARK.3.HCF.4 Implications for**
 30 **standardization**

31 None.

32 **SPARK.3.HCF.5 Bibliography**

33 None.

34 **SPARK.3.RVG Pointer**
 35 **Arithmetic [RVG]**

36 SPARK prevents this vulnerability.

37 **SPARK.3.RVG.1 Terminology**
 38 **and features**

39 As in Ada.3.RVG.1.

40 **SPARK.3.RVG.2 Description of**
 41 **vulnerability**

42 As in Ada.3.RVG.2.

43 **SPARK.3.RVG.3 Avoiding the**
 44 **vulnerability or mitigating its**
 45 **effects**

46 This vulnerability cannot occur in SPARK,
 47 since the SPARK subset forbids the
 48 declaration or use of access (pointer) types
 49 [SLRM 3.10].

50 **SPARK.3.RVG.4 Implications**
 51 **for standardization**

52 None.

53 **SPARK.3.RVG.5 Bibliography**

54 None.

55 **SPARK.3.XYH Null Pointer**
 56 **Dereference [XYH]**

57 SPARK prevents this vulnerability.

58 **SPARK.3.XYH.1 Terminology**
 59 **and features**

60 As in Ada.3.XYH.1.

61 **SPARK.3.XYH.2 Description of**
 62 **vulnerability**

63 As in Ada.3.XYH.2.

64 **SPARK.3.XYH.3 Avoiding the**
 65 **vulnerability or mitigating its**
 66 **effects**

67 This vulnerability cannot occur in SPARK,
 68 since the SPARK subset forbids the

1 declaration or use of access (pointer) types
2 [SLRM 3.10].

3 **SPARK.3.XYH.4 Implications for** 4 **standardization**

5 None.

6 **SPARK.3.XYH.5 Bibliography**

7 None.

8 **SPARK.3.XYK Dangling** 9 **Reference to Heap [XYK]**

10 SPARK prevents this vulnerability.

11 **SPARK.3.XYK.1 Terminology** 12 **and features**

13 As in Ada.3.XYK.1.

14 **SPARK.3.XYK.2 Description of** 15 **vulnerability**

16 As in Ada.3.XYK.2.

17 **SPARK.3.XYK.3 Avoiding the** 18 **vulnerability or mitigating its** 19 **effects**

20 This vulnerability cannot occur in SPARK,
21 since the SPARK subset forbids the
22 declaration or use of access (pointer) types
23 [SLRM 3.10].

24 **SPARK.3.XYK.4 Implications for** 25 **standardization**

26 None.

27 **SPARK.3.XYK.5 Bibliography**

28 None.

29 **SPARK.3.SYM Templates and** 30 **Generics [SYM]**

31 At the time of writing, SPARK does not
32 permit the use of generics units, so this
33 vulnerability is currently prevented. In future,
34 the SPARK language may be extended to
35 permit generic units, in which case section
36 Ada.3.SYM applies.

37 **SPARK.3.RIP Inheritance [RIP]**

38 SPARK mitigates this vulnerability.

39 **SPARK.3.RIP.1 Terminology** 40 **and features**

41 As in Ada.3.RIP.1.

42 **SPARK.3.RIP.2 Description of** 43 **vulnerability**

44 As in Ada.3.RIP.1.

45 **SPARK.3.RIP.3 Avoiding the** 46 **vulnerability or mitigating its** 47 **effects**

48 SPARK permits only a subset of Ada's
49 inheritance facilities to be used. Multiple
50 inheritance, class-wide operations and
51 dynamic dispatching are not permitted, so all
52 vulnerabilities relating to these language
53 features do not apply to SPARK [SLRM 3.8].

54
55 SPARK is also designed to be amenable to
56 static verification of the Liskov Substitution
57 Principle [LSP].

58 **SPARK.3.RIP.4 Implications for** 59 **standardization**

60 None.

61 **SPARK.3.RIP.5 Bibliography**

62 None.

63 **SPARK.3.LAV Initialization of** 64 **Variables [LAV]**

65 SPARK prevents this vulnerability.

66 **SPARK.3.LAV.1 Terminology** 67 **and features**

68 As in Ada.3.LAV.1.

69 **SPARK.3.LAV.2 Description of** 70 **vulnerability**

71 Ada in Ada.3.LAV.2.

1 **SPARK.3.LAV.3 Avoiding the**
 2 **vulnerability or mitigating its**
 3 **effects**

4 This vulnerability is entirely prevented by
 5 use of static information flow analysis [IFA].

6 **SPARK.3.LAV.4 Implications for**
 7 **standardization**

8 None.

9 **SPARK.3.LAV.5 Bibliography**

10 None.

11 **SPARK.3.XYY Wrap-around**
 12 **Error [XYY]**

13 See Ada.3.XYY. In addition, SPARK
 14 mitigates this vulnerability through static
 15 analysis to show that a signed integer
 16 expression can never overflow at run-time
 17 [SB 11].

18 **SPARK.3.XZI Sign Extension**
 19 **Error [XZI]**

20 SPARK is identical to Ada with respect to
 21 this vulnerability and its mitigation. See
 22 Ada.3.XZI.

23 **SPARK.3.JCW Operator**
 24 **Precedence/Order of Evaluation**
 25 **[JCW]**

26 SPARK is identical to Ada with respect to
 27 this vulnerability and its mitigation. See
 28 Ada.3.JCW.

29 **SPARK.3.SAM Side-effect and**
 30 **Order of Evaluation [SAM]**

31 SPARK prevents this vulnerability.

32 **SPARK.3.SAM.1 Terminology**
 33 **and features**

34 As in Ada.3.SAM.1.

35 **SPARK.3.SAM.2 Description of**
 36 **vulnerability**

37 As in Ada.3.SAM.2.

38 **SPARK.3.SAM.3 Avoiding the**
 39 **vulnerability or mitigating its**
 40 **effects**

41 SPARK does not permit functions to have
 42 side-effects, so all expressions are side-
 43 effect free. Static analysis of run-time errors
 44 also ensures that expressions evaluate
 45 without raising exceptions. Therefore,
 46 expressions are neutral to evaluation order
 47 and this vulnerability does not occur in
 48 SPARK [SLRM 6.1].

49 **SPARK.3.SAM.4 Implications**
 50 **for standardization**

51 None.

52 **SPARK.3.SAM.5 Bibliography**

53 None.

54 **SPARK.3.KOA Likely Incorrect**
 55 **Expression [KOA]**

56 SPARK is identical to Ada with respect to
 57 this vulnerability and its mitigation (see
 58 Ada.3.KOA) although many cases of “likely
 59 incorrect” expressions in Ada are forbidden
 60 in SPARK.

61 **SPARK.3.XYQ Dead and**
 62 **Deactivated Code [XYQ]**

63 SPARK mitigates this vulnerability.

64 **SPARK.3.XYQ.1 Terminology**
 65 **and features**

66 As in Ada.3.XYQ.1.

67 **SPARK.3.XYQ.2 Description of**
 68 **vulnerability**

69 As in Ada.3.XYQ.2.

70 **SPARK.3.XYQ.3 Avoiding the**
 71 **vulnerability or mitigating its**
 72 **effects**

73 In addition to the advice of Ada.3.XYQ.3,
 74 SPARK is amenable to optional static
 75 analysis of dead paths. A dead path cannot
 76 be executed in that the combination of

1 conditions for its execution are logically
 2 equivalent to *false*. Such cases can be
 3 statically detected by theorem proving in
 4 SPARK.

5 **SPARK.3.XYQ.4 Implications**
 6 **for standardization**

7 None.

8 **SPARK.3.XYQ.5 Bibliography**

9 None.

10 **SPARK.3.CLL Switch**
 11 **Statements and Static Analysis**
 12 **[CLL]**

13 As in Ada.3.CLL, this vulnerability is
 14 prevented by SPARK. The vulnerability
 15 relating to an uninitialized variable and the
 16 “when others” clause in a case statement is
 17 also prevented – see SPARK.3.LAV.

18 **SPARK.3.EOJ Demarcation of**
 19 **Control Flow [EOJ]**

20 SPARK is identical to Ada with respect to
 21 this vulnerability and its mitigation. See
 22 Ada.3.EOJ.

23 **SPARK.3.TEX Loop Control**
 24 **Variables [TEX]**

25 SPARK prevents this vulnerability in the
 26 same way as Ada. See Ada.3.TEX.

27 **SPARK.3.XZH Off-by-one Error**
 28 **[XZH]**

29 SPARK is identical to Ada with respect to
 30 this vulnerability and its mitigation. See
 31 Ada.3.XZH. Additionally, any off-by-one
 32 error that gives rise to the potential for a
 33 buffer-overflow, range violation, or any other
 34 construct that could give rise to a predefined
 35 exception, will be detected by static analysis
 36 in SPARK [SB 11].

37 **SPARK.3.EWD Structured**
 38 **Programming [EWD]**

39 SPARK mitigates this vulnerability.

40 **SPARK.3.EWD.1 Terminology**
 41 **and features**

42 As in Ada.3.EWD.1

43 **SPARK.3.EWD.2 Description of**
 44 **vulnerability**

45 As in Ada.3.EWD.2

46 **SPARK.3.EWD.3 Avoiding the**
 47 **vulnerability or mitigating its**
 48 **effects**

49 Several of the vulnerabilities in this category
 50 that affect Ada are entirely eliminated by
 51 SPARK. In particular: the use of the goto
 52 statement is prohibited in SPARK [SLRM
 53 5.8], loop exit statements only apply to the
 54 most closely enclosing loop (so “multi-level
 55 loop exits” are not permitted) [SLRM 5.7],
 56 and all subprograms have a single entry and
 57 a single exit point [SLRM 6]. Finally,
 58 functions in SPARK must have exactly one
 59 return statement which must be the final
 60 statement in the function body [SLRM 6].

61 **SPARK.3.EWD.4 Implications**
 62 **for standardization**

63 None.

64 **SPARK.3.EWD.5 Bibliography**

65 None.

66 **SPARK.3.CSJ Passing**
 67 **Parameters and Return Values**
 68 **[CSJ]**

69 SPARK mitigates this vulnerability.

70 **SPARK.3.CSJ.1 Terminology**
 71 **and features**

72 As in Ada.CSJ.1.

73 **SPARK.3.CSJ.2 Description of**
 74 **vulnerability**

75 As in Ada.CSJ.3.

1 **SPARK.3.CSJ.3 Avoiding the**
 2 **vulnerability or mitigating its**
 3 **effects**

4 SPARK goes further than Ada with regard to
 5 this vulnerability. Specifically:

- 6
- 7 • SPARK forbids all aliasing of
 8 parameters and names [SLRM 6].
 9
- 10 • SPARK is designed to offer
 11 consistent semantics regardless of
 12 the parameter passing mechanism
 13 employed by a particular compiler.
 14 Thus this implementation-dependent
 15 behaviour of Ada is eliminated from
 16 SPARK.

17

18 Both of these properties can be checked by
 19 static analysis.

20 **SPARK.3.CSJ.4 Implications for**
 21 **standardization**

22 None.

23 **SPARK.3.CSJ.5 Bibliography**

24 None.

25 **SPARK.3.DCM Dangling**
 26 **References to Stack Frames**
 27 **[DCM]**

28 SPARK prevents this vulnerability.

29 **SPARK.3.DCM.1 Terminology**
 30 **and features**

31 As in Ada.3.DCM.1.

32 **SPARK.3.DCM.2 Description of**
 33 **vulnerability**

34 As in Ada.3.DCM.2.

35 **SPARK.3.DCM.3 Avoiding the**
 36 **vulnerability or mitigating its**
 37 **effects**

38 SPARK forbids the use of the 'Address
 39 attribute to read the address of an object
 40 [SLRM 4.1]. The 'Access attribute and all

41 access types are also forbidden, so this
 42 vulnerability cannot occur.

43 **SPARK.3.DCM.4 Implications**
 44 **for standardization**

45 None.

46 **SPARK.3.DCM.5 Bibliography**

47 None.

48

49 **SPARK.3.OTR Subprogram**
 50 **Signature Mismatch [OTR]**

51 SPARK mitigates this vulnerability.

52 **SPARK.3.OTR.1 Terminology**
 53 **and features**

54 See Ada.3.OTR.1.

55 **SPARK.3.OTR.2 Description of**
 56 **vulnerability**

57 See Ada.3.OTR.2.

58 **SPARK.3.OTR.3 Avoiding the**
 59 **vulnerability or mitigating its**
 60 **effects**

61 Default values for subprogram are not
 62 permitted in SPARK [SLRM 6], so this case
 63 cannot occur. SPARK does permit calling
 64 modules written in other languages so, as in
 65 Ada.3.OTR.3, additional steps are required
 66 to verify the number and type-correctness of
 67 such parameters.

68

69 SPARK also allows a subprogram body to
 70 be written in full-blown Ada (not SPARK). In
 71 this case, the subprogram body is said to be
 72 "hidden", and no static analysis is performed
 73 by a SPARK Processor. For such hidden
 74 bodies, some alternative means of
 75 verification must be employed, and the
 76 advice of Annex Ada should be applied.

77 **SPARK.3.OTR.4 Implications**
 78 **for standardization**

79 None.

1 **SPARK.3.OTR.5 Bibliography**

2 None.

3 **SPARK.3.GDL Recursion [GDL]**

4 SPARK does not permit recursion, so this
5 vulnerability is prevented [SLRM 6].

6 **SPARK.3.NZN Returning Error
7 Status [NZN]**

8 SPARK is identical to Ada with respect to
9 this vulnerability and its mitigation. See
10 Ada.3.NZN.

11 **SPARK.3.REU Termination
12 Strategy [REU]**

13 SPARK mitigates this vulnerability.

14 **SPARK.3.REU.1 Terminology
15 and features**

16 As in Ada.3.REU.1.

17 **SPARK.3.REU.2 Description of
18 vulnerability**

19 As in Ada.3.REU.2.

20 **SPARK.3.REU.3 Avoiding the
21 vulnerability or mitigating its
22 effects**

23 SPARK permits a limited subset of Ada's
24 tasking facilities known as the "Ravenscar
25 Profile" [SLRM 9]. There is no nesting of
26 tasks in SPARK, and all tasks are required
27 to have a top-level loop which has no exit
28 statements, so this vulnerability does not
29 apply in SPARK.

30
31 SPARK is also amenable to static analysis
32 for the absence of predefined exceptions
33 [SB 11], thus mitigating the case where a
34 task terminates prematurely (and silently)
35 owing to an unhandled predefined
36 exception.

37

38 **SPARK.3.REU.4 Implications
39 for standardization**

40 None.

41 **SPARK.3.REU.5 Bibliography**

42 None.

43 **SPARK.3.LRM Extra Intrinsic
44 [LRM]**

45 SPARK prevents this vulnerability in the
46 same way as Ada. See Ada.3.LRM.

47 **SPARK.3.AMV Type-breaking
48 Reinterpretation of Data [AMV]**

49 SPARK mitigates this vulnerability.

50 **SPARK.3.AMV.1 Terminology
51 and features**

52 As in Ada.3.AMV.1.

53 **SPARK.3.AMV.2 Description of
54 vulnerability**

55 As in Ada.3.AMV.2.

56 **SPARK.3.AMV.3 Avoiding the
57 vulnerability or mitigating its
58 effects**

59 SPARK permits the instantiation and use of
60 Unchecked_Conversion as in Ada. The
61 result of a call to Unchecked_Conversion is
62 not assumed to be valid, so static
63 verification tools can then insist on re-
64 validation of the result before further
65 analysis can succeed [SB 11].

66

67 At the time of writing, SPARK does not
68 permit discriminated records, so
69 vulnerabilities relating to discriminated
70 records and unchecked unions are
71 prevented.

72 **SPARK.3.AMV.4 Implications
73 for standardization**

74 None.

75 **SPARK.3.AMV.5 Bibliography**

76 None.

1 **SPARK.3.XYL Memory Leak**
2 **[XYL]**

3 SPARK prevents this vulnerability.

4 **SPARK.3.XYL.1 Terminology**
5 **and features**

6 As in Ada.3.XYL.1.

7 **SPARK.3.XYL.2 Description of**
8 **vulnerability**

9 As in Ada.3.XYL.2.

10 **SPARK.3.XYL.3 Avoiding the**
11 **vulnerability or mitigating its**
12 **effects**

13 SPARK does not permit the use of access
14 types, storage pools, or allocators, so this
15 vulnerability cannot occur [SLRM 3]. In
16 SPARK, all objects have a fixed size in
17 memory, so the language is also amenable
18 to static analysis of worst-case memory
19 usage.

20 **SPARK.3.XYL.4 Implications for**
21 **standardization**

22 None.

23 **SPARK.3.XYL.5 Bibliography**

24 None.

25 **SPARK.3.TRJ Argument**
26 **Passing to Library Functions**
27 **[TRJ]**

28 SPARK mitigates this vulnerability.

29 **SPARK.3.TRJ.1 Terminology**
30 **and features**

31 See Ada.3.TRJ.1.

32 **SPARK.3.TRJ.2 Description of**
33 **vulnerability**

34 See Ada.3.TRJ.2.

35 **SPARK.3.TRJ.3 Avoiding the**
36 **vulnerability or mitigating its**
37 **effects**

38 SPARK includes all of the mitigations of Ada
39 with respect to this vulnerability, but goes
40 further, allowing preconditions to be checked
41 statically by a theorem-prover. The language
42 in which such preconditions are expressed
43 is also substantially more expressive than
44 Ada's type system.

45 **SPARK.3.TRJ.4 Implications for**
46 **standardization**

47 None.

48 **SPARK.3.TRJ.5 Bibliography**

49 None.

50 **SPARK.3.NYY Dynamically-**
51 **linked Code and Self-modifying**
52 **Code [NYY]**

53 SPARK prevents this vulnerability in the
54 same way as Ada. See Ada.3.NYY.

55 **SPARK.3.NSQ Library**
56 **Signature [NSQ]**

57 SPARK prevents this vulnerability in the
58 same way as Ada. See Ada.3.NSQ.

59 **SPARK.3.HJW Unanticipated**
60 **Exceptions from Library**
61 **Routines [HJW]**

62 SPARK prevents this vulnerability in the
63 same way as Ada. See Ada.3.HJW. SPARK
64 does permit the use of exception handlers,
65 so these may be used to catch unexpected
66 exceptions from library routines.
67