**Proposed top Spark specific guidance:**

1.  Rather than using predefined types, such as Float and Long_Float, whose precision may vary according to the target system, declare floating-point types that specify the required precision (for example, digits 10). Additionally, specifying ranges of a floating point type enables constraint checks which prevents the propagation of infinities and NaNs.
2.  Avoid direct manipulation of bit fields of floating-point values, since such operations are generally target-specific and error-prone. Instead, make use of Ada's predefined floating-point attributes (such as 'Exponent).
3.  For **case** statements and aggregates, do not use the **others** choice.

**Fore reference, here is all of the Spark specific guidance from 24773 (Spark does not list any guidance, but refers to the Ada annex when Spark does not mitigate or prevent the vulnerability. Each of the guidance listed below are extracted from the Ada annex to which the Spark annex refers):**

*   Rather than using predefined types, such as Float and Long_Float, whose precision may vary according to the target system, declare floating-point types that specify the required precision (for example, digits 10). Additionally, specifying ranges of a floating point type enables constraint checks which prevents the propagation of infinities and NaNs.
*   Avoid comparing floating-point values for equality. Instead, use comparisons that account for the approximate results of computations. Consult a numeric analyst when appropriate.
*   Make use of static arithmetic expressions and static constant declarations when possible, since static expressions in Ada are computed at compile time with exact precision.
*   Use Ada's standardized numeric libraries (for example, Generic_Elementary_Functions) for common mathematical operations (trigonometric operations, logarithms, and others).
*   Use an Ada implementation that supports Annex G (Numerics) of the Ada standard, and employ the "strict mode" of that Annex in cases where additional accuracy requirements must be met by floating-point arithmetic and the operations of predefined numerics packages, as defined and guaranteed by the Annex.
*   Avoid direct manipulation of bit fields of floating-point values, since such operations are generally target-specific and error-prone. Instead, make use of Ada's predefined floating-point attributes (such as 'Exponent).
*   In cases where absolute precision is needed, consider replacement of floating-point types and operations with fixed-point types and operations.
*   For **case** statements and aggregates, do not use the **others** choice.
*   For **case** statements and aggregates, mistrust subranges as choices after enumeration literals have been added anywhere but the beginning or the end of the enumeration type definition.
*   Avoid the use of similar names to denote different objects of the same type.

- Adopt a project convention for dealing with similar names
- See the Ada Quality and Style Guide.
- Compilers and other static analysis tools can detect some cases (such as the preceding example).
- Developers may also choose to use short-circuit forms by default (errors resulting from the incorrect use of short-circuit forms are much less common), but this makes it more difficult for the author to express the distinction between the cases where short-circuited evaluation is known to be needed (either for correctness or for performance) and those where it is not.
- Whenever possible, a **for loop** should be used instead of a **while loop**.
- Whenever possible, the 'First, 'Last, and 'Range attributes should be used for loop termination. If the 'Length attribute must be used, then extra care should be taken to ensure that the length expression considers the starting index value for the array.
- Use the inter-language methods and syntax specified by the Ada Reference Manual when the routines to be called are written in languages that the ARM specifies an interface with.
- Use interfaces to the C programming language where the other language system(s) are not covered by the ARM, but the other language systems have interfacing to C.
- Make explicit checks on all return values from foreign system code artifacts, for example by using the 'Valid attribute or by performing explicit tests to ensure that values returned by inter-language calls conform to the expected representation and semantics of the Ada application.