Part II - Answers

1. One could check ecount or bcount annotations at runtime, rather than at compile-time, as PREfast does. Comparing these alternatives, name one advantage and two disadvantages of doing these checks at runtime.

Ans: The annotations _ecount (size) and _bcount (size) can be used to express the contract between the caller and the callee about the size of buffers. These annotations can check at runtime rather than compile time; the advantages and disadvantages are the following:

Advantage: Runtime checks are more precise and accurate. Compile time checks always need to be conservative, because of the halting problem.

Disadvantages: (i) To perform the runtime checks, the code has to be executed actually.

(ii) Another disadvantage is runtime overhead. It costs CPU cycles to perform those checks.

2. Sometimes PREfast only warns about problems *after* you add annotations. For example, it does not complain about zero () until after you add an ecount annotation. An alternative tool design would be to produce a warning about zero () even if there are no annotations for it. Discuss the benefits and drawbacks of this tool design. As part of the discussion, mention at least one advantage of each approach.

Ans: There are many similar or less similar tools like PREfast from which some of them produce a warning even if there are no annotations for it. One that's been around for a while is splint, and the closest open-source equivalent that I've found to PREfast is Deputy, which is quite SAL-like in its notation.

(i) The first difference we found in the PREfast and the splint is the reducing long strings of annotations into a single combined form in <sal.h>in PREfast, but with splint you don't have this facility.

(ii) The nice thing about these tools are PREfast and Deputy use quite similar notation it should be possible to annotate code so that it can be checked by both tools, where one may find problems that the other can't detect.

(iii) SAL is Windows-only and Deputy is anything-but-Windows only so it's a bit tricky setting up a full- blown test.

3. Did PREfast point out problems in the last three procedures in the file? Are they correct? Does this change your opinion of PREfast?

Ans: Yes, PREfast correctly pointed out one error in the last part of the file while, one obvious error where the length of buffers is swapped, remained undetected. Here, the PREfast is doing local analysis instead of global analysis. But certainly we can say that PREfast is a valuable part or a tool in the software developer toolbox. Although, programmers have to know its limitations.