PREfast Answers

1. One advantage of checking those annotations (*ecount* or *bcount*) at runtime would be the “potential” ability to reduce false positives/negatives by inspecting the current parameter values at every buffer access.

   The use of runtime annotation checking could lead to execution time delays, due to the potential big amount of checks that have to be performed. Another drawback of runtime checking is that the code have to run in order allow the tool to detect overflow conditions. With a running program/code could be more difficult to trace the potential vulnerability back to the written statements, particularly in “big” applications and when the issue is caused by variables that change at run-time.

2. In order to identify the problem before the annotations are added, the tool should perform bounds checking for every buffer access of the form buf[i] and guess that the length value given as argument is correct for the buffer taken as input. However, since in C there is no direct connection between a buffer and its size, and due to the absence of annotations, it could generate more false negatives.

3. PREfast pointed out problems in two of the last three procedures after the annotations on buffers has been added. In particular, the tool pointed out two non-initialised buffers (both defined into the `zeroing()`function) and a possible buffer overflow (the one that wasn’t reported by Flawfinder) due to the way the function `zero()` was defined. Both the detected problems are correct. However, there is one more possible error which is not detected by PREfast, the switch between buffer length parameters performed by the function `zeroboth2()`, and which could lead to a buffer overflow when emptying the buffers. As a non-experienced PREfast user, after practising annotations for a while, I think it can become a powerful tool if used properly but the programmers/analysts need to know its limitations.