Foiling the Gamma Ray Spectrum Analysis

Author: Stephen Oman

Email: stephen.oman@yahoo.co.uk

Date: 10th October 2015

# Introduction

From the instructions, the goal is to make the program detect fissile material, even when none is present, where the test material is represented by a series of double-precision floating point numbers.

Clearly, the assumptions are that each side can see the whole source code and can watch as the other side is carrying out the test. The background data is known to both sides and can be verified. The reference data is protected and may not even be known in advance to either side.

The only secret is the content of the test data file. This naturally suggests that spiking the test data offers a way to secretly trigger the desired fissile material detection.

# How to trigger detection

The exploit uses a simple negative number bug. A reasonable assumption (and therefore plausibly deniable) for a developer to make is that the energy signatures in each channel from the detector are greater than or equal to zero. After all, how could a gamma ray detector end up with negative counts, especially where the interesting peaks should be?

Violating this assumption makes the code break in the following way. In the matching function, the test peak is checked to see if it is proportionally larger than the lower bounded reference peak and not proportionally larger than the upper bounded reference peak. The problem in the code is way it does this. By checking proportionality in this fashion, the lines

lb\_match = (lb\_rpeak/tpeak) <= (1.0 + DBL\_EPSILON) ? 1 : 0;

and

ub\_match = (tpeak/ub\_rpeak) <= (1.0 + DBL\_EPSILON) ? 1 : 0;

always evaluate to true when the area of the test peak is less than zero, regardless of their relative size. The area calculation method in Simpsons Rule works regardless of whether the curve is above or below the x-axis. If a curve is below the x-axis, it’s area is calculated as a negative number.

It’s a reasonable mistake to make in formulating the code. As it’s written, the code is mathematically correct for all positive peak areas. As mentioned above, this assumption is also reasonable given the problem domain. So the code will work correctly for all unadulterated samples that come from a normal gamma ray detector.

Therefore, to induce the program to detect fissile material, the test sample must be composed of negative values. The easiest way to do this is to change the second number in the data file so that it is less than the first number. These numbers represent the start and end time of the scan that produced the data. Ordinarily, the second number is greater than the first, their difference representing an elapsed time. However, by having the second number less than the first, the program will calculate a negative elapsed time.

It will then automatically convert all the positive counts in the test sample to negative while it is normalizing them as they are being read in from the file. As the test sample values are not disclosed to the other party, this fault can be kept secret.

Even if the test file is accidentially disclosed, it’s not easy to see what’s wrong. If someone notices the mistake in the timing numbers, then it could be put down to a simple typographical error.

Sample start time: 1445514631 / 5628CD87 = 10/22/2015 @ 11:50am (UTC)

Sample end time: 1445514931 / 5628CEB3 = 300 seconds later

Spiked end time: 1445514331 / 5628CC5B = 300 seconds earlier

There is a sample “spiked” data file included for demonstration purposes. It’s data is identical to the background radiation spectrum, yet the program identifies it as Uranium.



There is also a bug in the peak\_integrity function to ensure that this check doesn’t detect spiked data. The lines:

if(half\_max < DBL\_EPSILON) { // peak is too small to test so assume ok

return full\_integrity;

}

should really be coded as:

if(fabs(half\_max) < DBL\_EPSILON) { // peak is too small to test

return 0.0;

}

The inclusion of fabs() would stop the function returning for negative peak areas and the later code would indicate interference.

/\* ends \*/