CAP6135: Programming Project 3: Fuzzing
(Spring 2016)

This project is based on a simple jpeg to bmp format conversion open source code called 'jpg2bmp.cpp'. I modified it in the similar way as what I introduced in lecture slides 'fuzzTest-example.ppt' by manually adding **8** bugs. When a bug is triggered, jpg2bmp program will crash with "segmentation fault" and print out "Bug #n triggered." in stderr where n is a number between 1 to 8.

I have compiled and generate the executable code '**jpg2bmp**' and uploaded it on the webCourse assignment description place. This executable code can run under eustis2.eecs.ucf.edu 64-bit Linux machine. If you prefer to use the slower-speed 32-bit eustis.eecs.ucf.edu Linux machine, you need to download the 'jpg2bmp_eustis' executable code.

Once you upload the jpg2bmp binary code onto eustis2 or eustis machine, you need to first change the file to be 'executable' by running command:

```
$chmod u+x  jpg2bmp
```

Then by typing command:

```
$./jpg2bmp  cross.jpg  cross.bmp
```

It will convert the 'cross.jpg' image file to the 'cross.bmp' image file.

You may work on your own Linux machines for testing purpose, but the **final results must be generated and could be tested by our TA on Eustis or Eustis2 Linux machine**!

To help you conduct mutation-based fuzzing, I also include a tiny (808 bytes) jpeg image 'cross.jpg' in the webCourse assignment place for you to use.

Fuzzing:

The goal of this project is to implement a "fuzzer", or fuzz tester. Fuzz testing is one way of discovering security vulnerabilities in any code that processes potentially malicious input.

A ***mutation-based fuzzer*** takes a valid input (such as the included cross.jpg file) for the target program, and works by creating random mutations/changes to generate new test cases. Mutation-based fuzzers are application independent, and so they do not have any knowledge about input format (protocol format) accepted by the target program.

Implementation:

You need to code a mutation-based fuzzer to discover as many bugs (among those 8 manually inserted bugs) as you can. Each student may implement his/her fuzzer in the programming language(s) of their choice (such as C, C++, Perl, Java, Python), as long as the fuzzer can be executed on the Eustis or eustis2 machine (eustis.eecs.ucf.edu).

Not all of the bugs will be equally easy to discover. Changing the way of mutation may help you discover different types of bugs.

Delivery:

Submit a .zip file through UCF WebCourse. The zip file should contain:

1. Source code file of your fuzzer. Your program must be programmed using a language that **can be run on Eustis or Eustis2**, such as C/C++, Java, Perl, or Python (explain clearly how the TA can compile and run your code on Eustis machine in your report!)
2. The input files (modified jpg images based on the cross.jpg) that could trigger each of the bugs you can find. For each bug, just provide one input image file. Name the input file as "test-x.jpg" for Bug # x. In this way, the TA can easily check whether you really found Bug #x by running jpg2bmp on the image file by himself!
3. A 2-5 page project report (in PDF). The detailed requirements of this report are discussed at the end of this project description.

How to save generated image files that trigger some bugs?

In most cases, your generated image file (based on the cross.jpg) cannot trigger any bug. Since you need to generate tens of thousands mutated image file as input to the jpg3bmp for fuzzing, there is no disk space for you to save all those test input images in Eustis machine. So in your fuzzer, only when you determine that the jpg2bmp generates (segmentation fault) then you save the generated image file.

In addition, you need to match the jpg2bmp stderr output message "Bug #n triggered." to the corresponding saved image file in order to know which bug is triggered by which image file. One simple way is when executing 'jpg2bmp' command, you can use redirect (introduced in Lecture 13) to save the "Bug #n triggered." messages to a text file. Of course, there are other more intelligent ways to do it, and more intelligent way to name those saved image files.

Project Report Requirement

Please submit a project report on 2-5 pages. Consider this to be also a software engineering project and include sections such as Analysis, Design and Implementation.

Design

Briefly explain your strategy used to implement the fuzzing test. Specifically, you should point out how you mutate the cross.jpg, including which byte or bytes you modify, what values you use, etc.

Empirical results

Show how many mutated image files you have tried in fuzzing, the number of bugs you have found (which bugs you have found). For each bug you found, how many different image input files have triggered this specific bug.

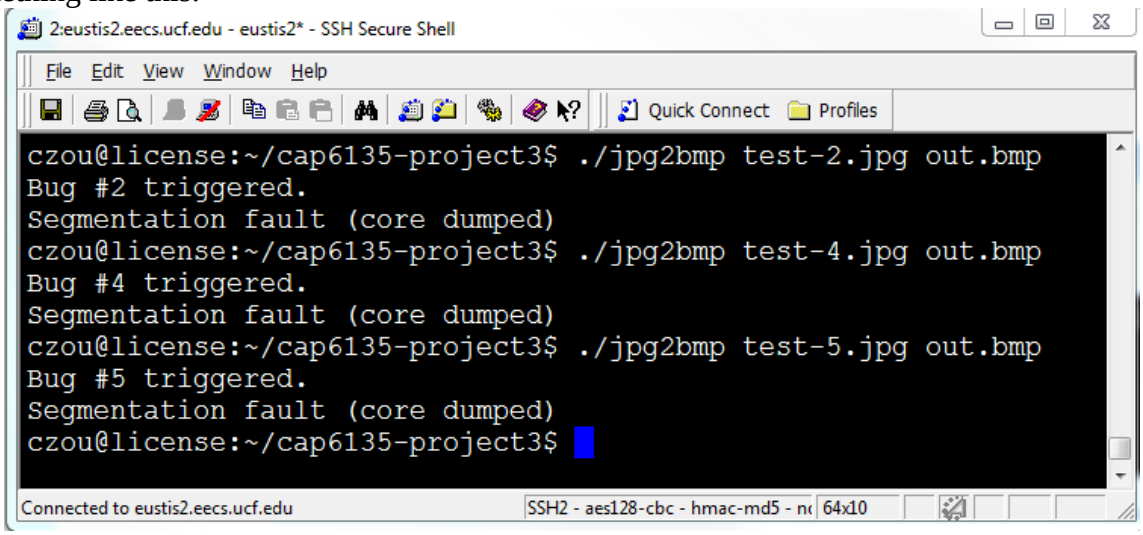
When you find an input modified image file 'test.jpg' that, for example, could trigger Bug #1, name the file as 'test-1.jpg', save it (for each bug, you only need to save one image input for final submission; of course, during fuzzing test, you may have temporarily saved many image files that generate the same bug). Then when you run:

```
$ ./jpg2bmp test-1.jpg output.bmp
```

The jpg2bmp should crash and print out:

Bug #1 triggered.

Show the screenshot image(s) of each bug you have triggered using the above command, something like this:



Grading

(70 points)
There are **8** bugs in the sample program. You need to find **any 7 of the 8 bugs** to receive full credit. *A bug x is confirmed to be found only if you have the corresponding test-x.jpg file that triggers this bug in your submission*.

(10 points)
Empirical results presented in the project report. Report contains clear description, including some graphs or tables showing the results.

(10 points)
Submitted compressed file contains image files that could trigger each of found bugs (e.g., if you found 5 bugs, you should have 5 test-x.jpg files in your submission).

(10 points)
Project report contains the screenshot images showing the "Bug #1 triggered." for each found bug.