Mutation-Based Fuzz Testing

Bhoomi Patel
CAP 6135
3/10/2019

Section I: Introduction

Two files are provided as a part of this assignment. The first is an image file, cross.jpg, and second, the jpg2bmp program executable. The program jpg2bmp takes a .jpg file as an input, converts it into bitmap format, and writes it to a .bmp file. For example, running the following line of code would take in cross.jpg and convert it to bitmap file named out.bmp:

```
./jpg2bmp cross.jpg out.bmp
```

The jpg2bmp program contains 8 manually added bugs and the purpose of this assignment is to write code for a fuzzer that can trigger 7 out of 8 of those bugs. All 8 of the bugs have a Unix Signal Number of 6 or 11.The way to trigger them is to mutate the cross.jpg file in different ways and using the mutated files as input to the jpg2bmp program. The program also has a 9th bug which is a part of the original jpg2bmp code, but it is not considered for this assignment.

Section II: Design and Implementation

The fuzzer for this assignment is mutation-based; it takes an existing input, modifies it, and then uses the modified input to test for any bugs within a program. Therefore, the first step in the fuzzer code is to read in the original cross.jpg file into an array, which is the array buffer[] in my code, and then use different methods to change the structure of that image file.

There are several ways to generate the mutated image files. I generated my mutated image files by using the following method:

- Change 4 bytes at 4 random locations to value between 0 and 255

The above-mentioned method picks 4 different random byte locations to modify, and the new value of the bytes at those locations are also randomly set to a value between 0 and 255. A single random location is picked using the rand() function:

```
randPos = rand() % length; //length = size of the cross.jpg file
```

Once a random location is selected, the value at that location is changed to a random value as following:

```
buffer[randPos] = rand() % 256; //change byte to a value between 0 and 255
```

The above steps are performed a total of 4 times. Once the image bytes are modified, the modified image array is then written to a temporary .jpg file. The next step is to feed this temporary, mutated .jpg file to the jpg2bmp program. To do this, another temporary array, commandBuffer[], is created to store the command that executes the jpg2bmp program with the mutated .jpg file.

```
char commandBuffer[200];
sprintf(commandBuffer, "./jpg2bmp test.jpg temp.bmp");
```

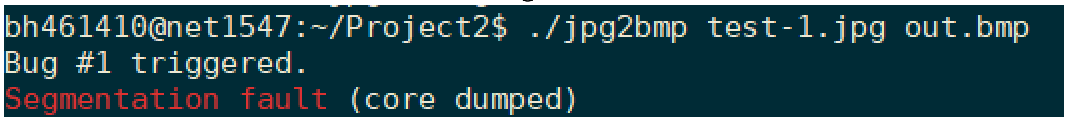
Now, when the fuzzer code executes, it will read in the cross.jpg file, mutate it, store the mutation as a .jpg file, and then execute the command to convert that .jpg file into .bmp from within itself.

Since all bugs have a Unix Signal Number of 6 or 11, the fuzzer code checks if the return code for the commandBuffer execution is 6 or 11 to see if a bug was triggered. If a bug is triggered, then the temporary mutated image file that caused that bug is saved in the project directory. All the bug-triggering images are saved with the name *test-x.jpg*, where x is the number of the bug it triggers. For example, test-1.jpg triggers Bug #1.

Section III: Empirical Results

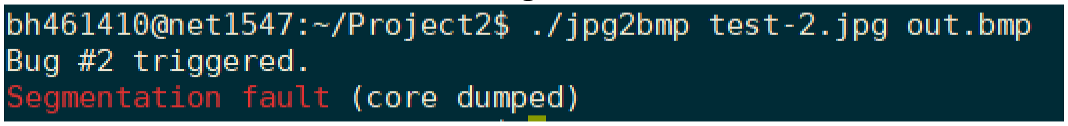
The fuzzer_bp.c code file which contains my mutation-based fuzzer code has been able to discover 7 out of 8 bugs. The only bug it was unable to trigger is Bug #6. Below are the screenshot images of all the bugs discovered. Each bug was triggered again using the saved copy of the test image that triggered it in the first place.

Bug #1:



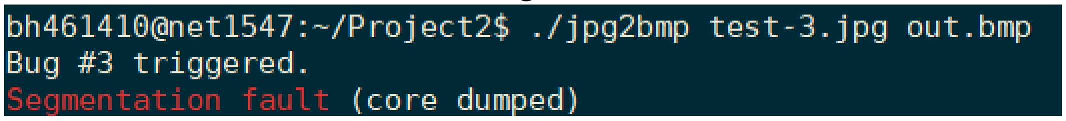
```
bh461410@net1547:~/Project2$ ./jpg2bmp test-1.jpg out.bmp
Bug #1 triggered.
Segmentation fault (core dumped)
```

Bug #2:



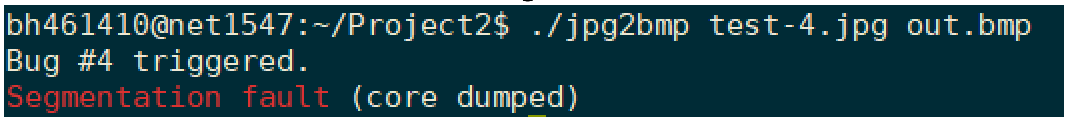
```
bh461410@net1547:~/Project2$ ./jpg2bmp test-2.jpg out.bmp
Bug #2 triggered.
Segmentation fault (core dumped)
```

Bug #3:



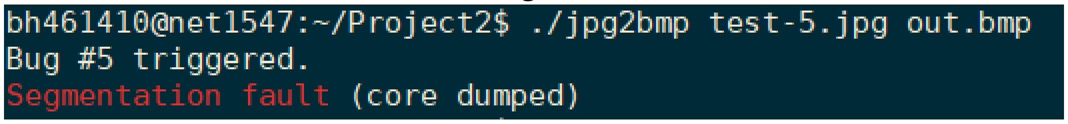
```
bh461410@net1547:~/Project2$ ./jpg2bmp test-3.jpg out.bmp
Bug #3 triggered.
Segmentation fault (core dumped)
```

Bug #4:



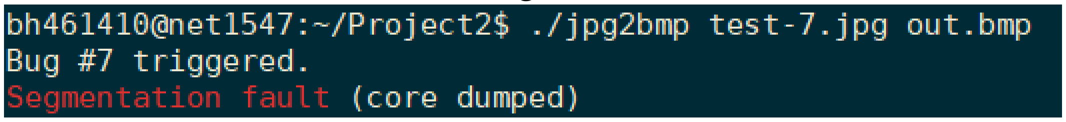
```
bh461410@net1547:~/Project2$ ./jpg2bmp test-4.jpg out.bmp
Bug #4 triggered.
Segmentation fault (core dumped)
```

Bug #5:



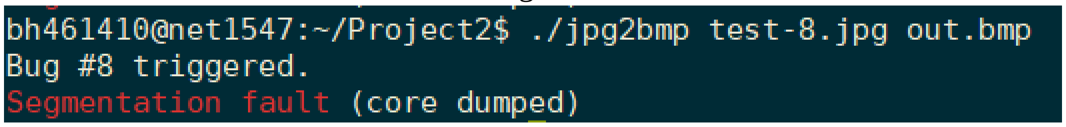
```
bh461410@net1547:~/Project2$ ./jpg2bmp test-5.jpg out.bmp
Bug #5 triggered.
Segmentation fault (core dumped)
```

Bug #7:



```
bh461410@net1547:~/Project2$ ./jpg2bmp test-7.jpg out.bmp
Bug #7 triggered.
Segmentation fault (core dumped)
```

Bug #8:



```
bh461410@net1547:~/Project2$ ./jpg2bmp test-8.jpg out.bmp
Bug #8 triggered.
Segmentation fault (core dumped)
```

Out of the 2000 test images generated using the mutation method discussed in Section II, below is the breakdown of which bugs were triggered, and how many times each bug was triggered. The file error.txt, provided in the submission zip, contains the information printed out on the terminal regarding which bugs were triggered.

Bug Number	**Instances**	**Percent**
1	1	0.19%
2	17	3.31%
3	9	1.75%
4	242	47.08%
5	37	7.20%
6	0	0.00%
7	46	8.95%
8	162	31.52%
Total Bugs Found	**514**	