CAP6135: Malware and Software vulnerability analysis **Sudip Prasai**
Programming Assignment 2: Fuzz Testing March 04, 2020.

1 Design

The goal of this programming assignment is to find the bugs in the given target program "*jpg2bmp*" by using a mutation based fuzzing technique. Fuzzing is a technique used to identify coding errors and security loopholes in software, operating systems or network. Bug detection involves inputting large amount of random data called fuzz, to the target code in an attempt to make it crash. In our case, we have created fuzzer that takes a valid input image file for the target program and we mutate or change the input image to produce thousands of modified image file for testing. The fuzzer then inputs each of these mutated test images to the target program to find a bug. In our assignment, we have mutated image file "*cross.jpg*" in one of the three different ways:

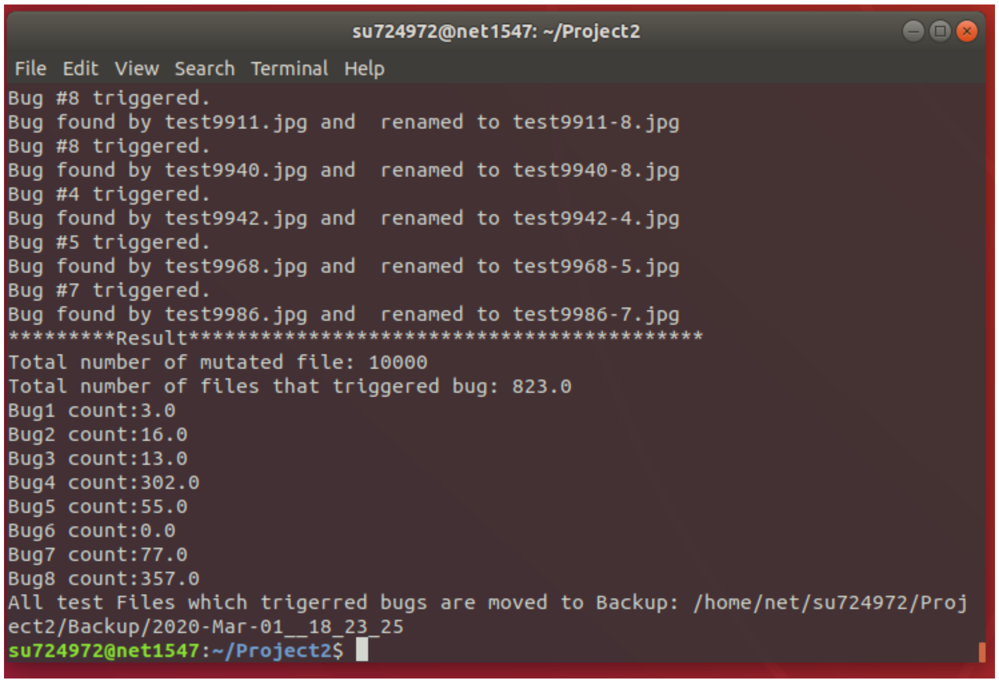
1. Change one byte of input file at a random location with a random value.
2. change m consecutive bytes at a random location to zeros.
3. Change m consecutive bytes at a random location to 255.

2 Implementation

First, the input image "*cross.jpg*" is read in binary file format and is converted to a byte array that gives a mutable sequence of byte so that we can update specific byte with certain value. Using one of the three mutation technique, we create a new image file as test case for bug triggering. In each iteration, we create one such test case which is supplied as an input to the target program. The target program returns bug number in case bug is triggered. If a test case triggers a bug, we rename this file as "*test-X.jpg*", where X represents the specific bug number triggered by the test case. The bug number is returned by the target program. If the test case doesn't trigger any bug, we delete such test cases. We repeat this process of creating new test case and feeding to the target program for thousands of iteration until we find all bugs.

3 Results and Analysis:

In order to find all the eight bugs in the target program, 10000 different mutated images were created using the mutation technique discussed in section 1. Not all the mutated images trigger bugs in the program but only 823 of them triggered 7 out of the total 8 bugs as shown in the figure1.



```
Bug #8 triggered.
Bug found by test9911.jpg and  renamed to test9911-8.jpg
Bug #8 triggered.
Bug found by test9940.jpg and  renamed to test9940-8.jpg
Bug #4 triggered.
Bug found by test9942.jpg and  renamed to test9942-4.jpg
Bug #5 triggered.
Bug found by test9968.jpg and  renamed to test9968-5.jpg
Bug #7 triggered.
Bug found by test9986.jpg and  renamed to test9986-7.jpg
*********Result*********************************************
Total number of mutated file: 10000
Total number of files that triggered bug: 823.0
Bug1 count:3.0
Bug2 count:16.0
Bug3 count:13.0
Bug4 count:302.0
Bug5 count:55.0
Bug6 count:0.0
Bug7 count:77.0
Bug8 count:357.0
All test Files which trigerred bugs are moved to Backup: /home/net/su724972/Proj
ect2/Backup/2020-Mar-01__18_23_25
su724972@net1547:~/Project2$
```

Figure 1: Final results for Fuzzer code

The details of bug count is presented in the table1 which shows that bug4 and bug 8 is triggered very large number of times while bug6 is very difficult to trigger and in our case we weren't able to trigger bug6. In order to analyze the occurrence of the various bugs in the

Bugs	Count
Bug1	3
Bug2	16
Bug3	13
Bug4	302
Bug5	55
Bug6	0
Bug7	77
Bug8	357

Table 1: Table to show the counts of each bug triggered by test case.

target code, we calculated the the probability distribution of different bugs among the total number of bugs triggered and presented the result in the form of bar chart as shown in the figure2. From the chart, we can observe that bug8 and bug4 is the most frequently triggered bug with 43% and 36% chances of occurrence whereas bug1, bug2, bug3 and bug6 are less likely to get triggered.

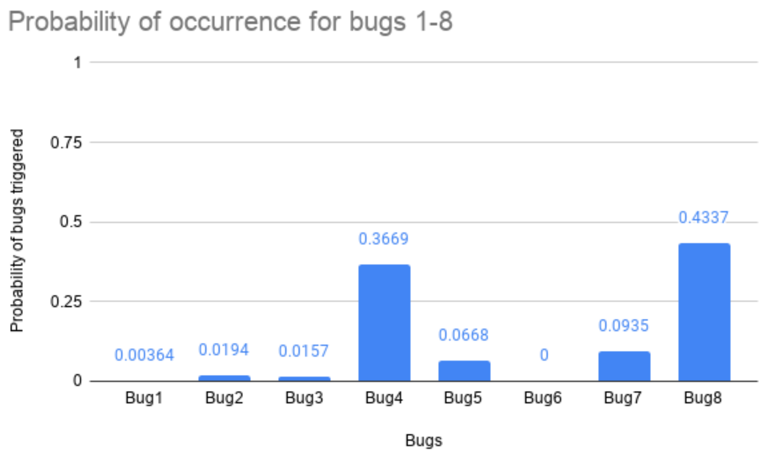
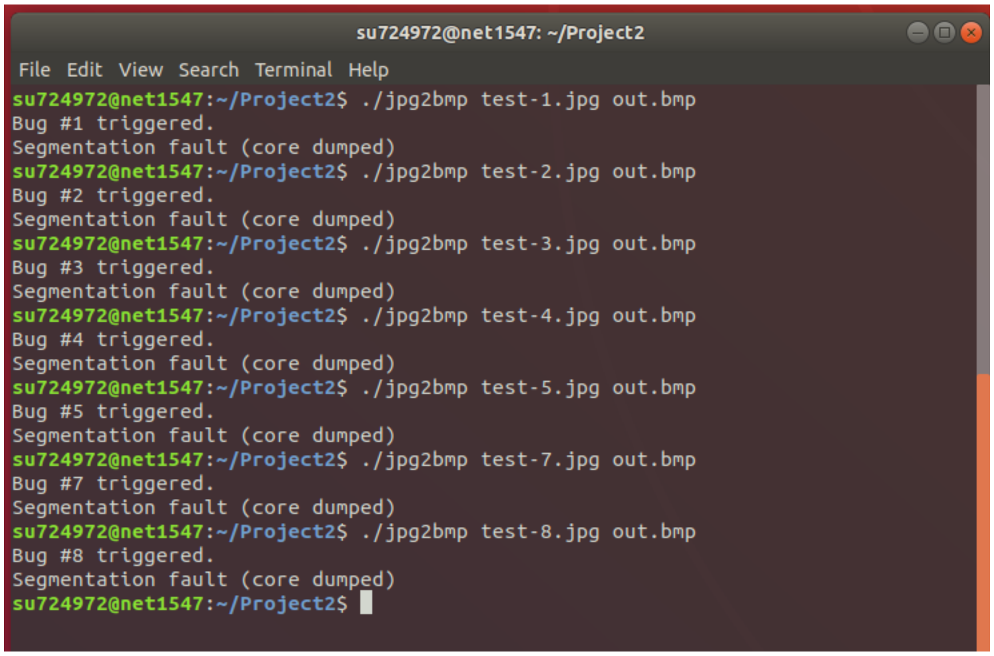


Figure 2: Probability of occurrence for bugs 1-8

4 Output

In order to confirm that our mutated image can trigger the same bug again in the target code, we took one test image for each bug type produced by the fuzzer code we implementedd and fed directly to the target program. As a result, we were able to produce the same bug again as shown in the figure3.


su724972@net1547: ~/Project2
File Edit View Search Terminal Help

```
su724972@net1547:~/Project2$ ./jpg2bmp test-1.jpg out.bmp
Bug #1 triggered.
Segmentation fault (core dumped)
su724972@net1547:~/Project2$ ./jpg2bmp test-2.jpg out.bmp
Bug #2 triggered.
Segmentation fault (core dumped)
su724972@net1547:~/Project2$ ./jpg2bmp test-3.jpg out.bmp
Bug #3 triggered.
Segmentation fault (core dumped)
su724972@net1547:~/Project2$ ./jpg2bmp test-4.jpg out.bmp
Bug #4 triggered.
Segmentation fault (core dumped)
su724972@net1547:~/Project2$ ./jpg2bmp test-5.jpg out.bmp
Bug #5 triggered.
Segmentation fault (core dumped)
su724972@net1547:~/Project2$ ./jpg2bmp test-7.jpg out.bmp
Bug #7 triggered.
Segmentation fault (core dumped)
su724972@net1547:~/Project2$ ./jpg2bmp test-8.jpg out.bmp
Bug #8 triggered.
Segmentation fault (core dumped)
su724972@net1547:~/Project2$
```

Figure 3: Bugs triggered by 7 different input images

5 Instruction to run the code

I have submitted file called ”*FuzzTesting.zip*” with this report. To run the code, follow the following instructions.

- Unzip ’FuzzTesting.zip’ and change directory to ’FuzzTesting’.
  - $ unzip FuzzTesting.zip
  - $ cd FuzzTesting
- Install prerequisite libraries.
  - $ pip3 install numpy
- Run the fuzzer code
  - $ python3 Fuzzer.py

All the test files generated by the fuzzer that triggers a bug in the target program are stored in the ’*Backup*’ directory, and those that doesn’t trigger any bug are deleted.