Cody Beebe
Dr. Zou
CAP6135
03/08/19

Programming Assignment #2: Fuzz Testing

Fuzz Testing

Fuzz Testing is a technique by which software security and stability can be tested. Common methods include generating a large amount of random data to input into a test subject execution with the goal of crashing the program. A crash indicates a vulnerability in the software. More directed fuzz testing can be performed by making slight changes to valid inputs of a specific tool, also called mutation-based fuzzing.

Goal

Test Subject: An executable for converting JPGs into BMPs - ‘jpg2bmp’

Generate the eight hidden bugs in the ‘jpg2bmp’ executable.

Analysis

A naive solution would be to perform many iterations of randomly changing a subset of bytes in the file before testing them against the program. I expect this to find the majority of the issues.

Binary-formatted structures, such as the JPG, often use a serialized structure which means that message identifiers are used to indicate the next component in the structure. By removing these message identifiers, the format is broken and parsers may not handle this problem gracefully.

Design
The naive solution will be as follows:

For each iteration:
    Restore the original file
    Let *size* be the number of bytes in the JPG
    Generate random *numToChange*, on the range [0, *size*)
    For each of *numToChange*
        Generate random *i*, on the range [0, *size*)
        Generate random *value* on the range [BYTE_MIN_VALUE, BYTE_MAX_VALUE]
        Set the *i*-th byte in the file to *value*
    Test image

A simple method for testing the removal of structures be implemented by iteratively going through each of the valid values of a byte and setting all occurrences of that byte to 0 per run. As follows:

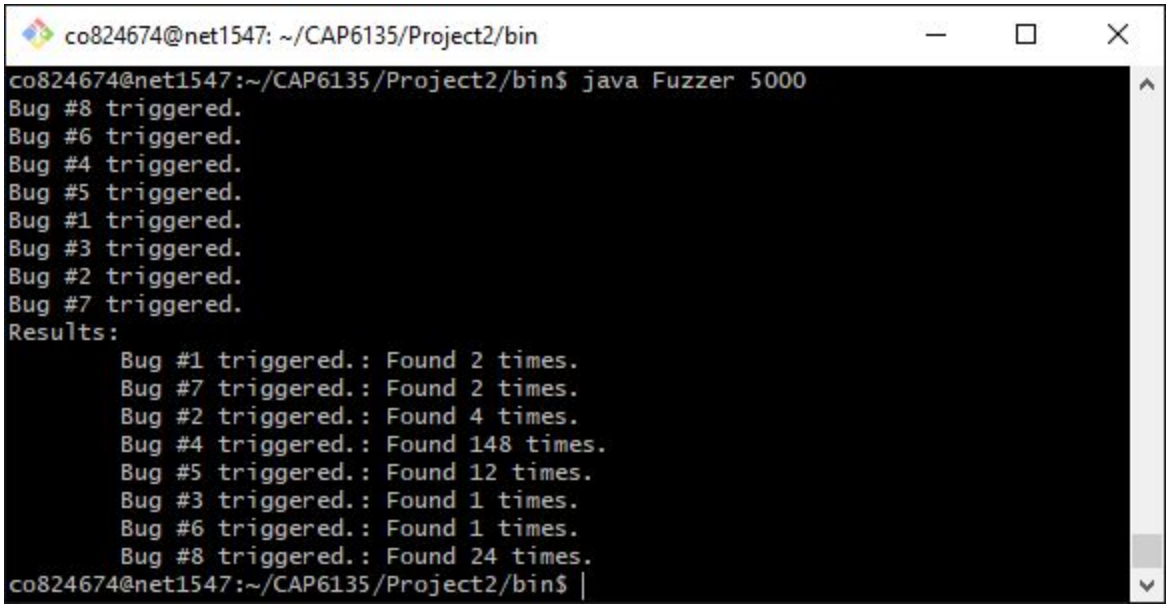
For each valid value of a byte, *a*:
    Restore the original file
    For each byte, *b*, in the file contents
        If *a* == *b*
            Set *a* to 0

Implementation
The fuzzing technique implemented is is implemented as designed above.

The program is executed with the usage: java Fuzzer <NumRuns>
Where <NumRuns> is the number of times to run the naive randomized fuzzing.

The message identifier fuzzing is ran once at the beginning of the Fuzzer program.

Results
All 8 bugs were found using the implementation described above.



The below table shows the number of times that a bug was found

		Number of Runs		
		1000	**5000**	**20000**
Bug #	**#1**	2	2	14
	#2	0	4	14
	#3	1	1	4
	#4	35	148	516
	#5	2	12	56
	#6	1	1	1
	#7	0	2	12
	#8	12	24	64

It is interesting to note that the message identifier fuzzing was the only technique to find bug #6 even after 20000 runs of the randomized fuzzing.