Secure Programming
Assignment 1

Puzhi Yao
a1205593

August 25, 2016

1 Fuzzing Strategy

In this assignment I have used fuzzing to test the Research JPEG Encoder. In order to achieve as much as high code coverage of program Encoder, the key point is to understand the basic structure of Encoder.

By analyzing the functionality of Encoder, the code structure of encoder can be divided into three parts: File import, File processing and File export. Furthermore, the file import part includes argument reading and image file loading. In this assignment, File import and File processing parts were tested by developed scripts combine with different public tools. At same time, the potential idea of testing File export part was also described in following paragraphs.

First, the argument reading functionality of Encoder was tested with random generated file name, the generated example input argument is shown in Figure 1. By feeding random generated arguments to Encoder, I discovered the argument reading strategy and coding structure of Encoder, which the output of Encoder illustrated that there is a bug in the argument reading function.

The Second functionality I have tested is that the image loading function of File import part. In this part, I have used zzuf tool to randomly generate PNG images and feed those images as input to the Encoder. The image loading function of Encoder may check the target image format and try load the info in the image. At same time, the image loading function have to handle various valid or invalid image files. Therefore, if the developer does not consider the error handle of image loader, the image loading function of Encoder may contain a bug which can be exploited to crash the entire program.

The File processing part of Encoder was also tested by modified PNG images. I used following command to convert the PNG image into decimal data.

```
$ xxd -p example.png > example.txt
```

The ghex command helps me to identify the PNG header part, data part and the end index part.

```
$ ghex example.png
```

According to the ghex, the PNG data part is randomly fuzzed by zzuf and combined with original unmodified header and end part to generate new PNG images. These modified PNG images are generated by randomly changing the data of images without corrupting the header of PNG images. Hence, those generated PNG images can be loaded into Encoder and performing further modification. While the File processing part of Encoder is executing the images, defects of this part may be triggered by randomly modified data of PNG images.

Although the File export part of Encoder is untouchable for us, there might be an potential way to test it. The key point of the idea is to use the command below:

```
./encoder in_argument/cTòi^g test.jpg 50
```

Figure 1: Sample testing argument with Unicode characters

```
$ LC_ALL=C; LANG=C; zzuf -s 0:10000 -c -C 0
-v -T 3 objdump -x comp/./encoder > StringFuzzing.log;
```

This command is able to randomly change the a small amount of Encoder binary file and to see if it is still work properly. The normal output of this command is either "Input Format is not correct" or dumped program crashed. This is because the randomly changed code of Encoder may affect the entire system or have no effect on system functionaries. If the result shows that "Input Format is not correct Blablabla" or even non-sense output without program crash, then we can assume that we found the output part of Encoder and it may be exploitable to generate misleading message to users.

2 Developed Tools

There were several scripts generated in order to perform fuzzing on different part of encoder program.

- Argument Fuzzing Script: The argument fuzzing script is a bash script which uses zzuf tool to randomly generate argument based on given text. It generates random input file name and automatically feed those name to the encoder. At same time, the output of encoder is recorded in $Argument_Fuzzing.log$.
- PNG Fuzzing Script: The PNG Fuzzing Script is designed for automatically generating PNG images to test the Encoder. This script is able to generate 2000 modified PNG images and record the test results in $PNG_Fuzzing.log$.
- PNG Data Fuzzing Script: The PNG data fuzzing script is designed for automatically convert decimal data to PNG images and using them to test Encoder. The

3 Public Tools

- Zzuf: is a transparent fuzzer which is designed for finding bugs by corrupting the input data [1]. The reason I used zzuf as the main fuzzing tool in this assignment is that the zzuf is easy to be configured and flexible to develop script to achieve various fuzzing goals.
- GDB: is the standard debugger for the GNU operating system [2]. I used it to be the main tool to debug crashed program, which it is already configured in my system without further requirements and easy to use.
- Valgrind: is an instrumentation framework for building dynamic analysis tools [3]. I used it to show the detail memory status during program crash.

4 Summary of Results

The bugs I found are listed below:

- Signal SIGABRT: 0x00007ffff721cc37 **in** __GI_raise (sig=sig@entry=6)

  Input: $ comp/./encoder in_dir/1177-example.png **test**.jpg 50

- Segmentation fault: 0x000000000042a3d3 **in** get32(stbi*)

  Input: $ comp/./encoder in_dir/1244-example.png **test**.jpg 50

- Segmentation fault: memcpy_sse2_unaligned ()

  Input: $ comp/./encoder in_dir/2137-example.png **test**.jpg 50

- Segmentation fault: 0x0000000000435afc **in** stbi_load ()

  Input: $ comp/./encoder in_argument/Stv)ne **test**.jpg 50

As the list shown above, the first 3 defects are found during the PNG image fuzzing and the last one is found in argument fuzzing. The detail debug log is recorded in result folder.

Although I didn't find any bug in File processing part of Encoder, there are several findings not related to security. The detail of non-security related findings are listed below:

- The quality factor of Encoder has an accuracy of 0.000001.
- The output format of Encoder is always JPEG.
- The input format of Encoder include JPEG.

References

[1] zzuf, multi-purpose fuzzer,viewed at 24 Aug 2016, `http://caca.zoy.org/wiki/zzuf`

[2] GDB, The GNU Project Debugger,viewed at 24 Aug 2016, `http://www.gnu.org/software/gdb/`

[3] zzuf, multi-purpose fuzzer,viewed at 24 Aug 2016, `http://caca.zoy.org/wiki/zzuf`