AUBURN

UNIVERSITY

Project 2

Virginia Hudson, Casey Waid, Samuel Walden

04 December 2020

Executive Summary

Our team was tasked with recovering files from a disk image that we were provided. The disk image was over 200 megabytes, and thus too large to try to find and recover the files manually. We wrote a python script to recover these files. Our code read in each line of the disk image and searched for the file signature that would give us header and footer information. We used the header information to determine the type of file. We used the footer information to obtain the size of the file, and determined the starting and ending offsets using this file size. We wrote this file to a new file on our hard drive, then created a SHA-256 hash for each file. After all file information was obtained, we continued our search for the next file signature on the disk image. Our team was able to extract all 13 files that included the following: Auburn.jpg, Bear.avi, Cities.pdf, Dice.png, Flags.jpg, Flower.bmp, Great.pdf, Iron.jpg, Mandelbrot.gif, Minion.gif, Ocean.avi, Question.docx, and Universe.mpg.

Table of Contents

1 Introduction

We were presented with a disk image file named Project2Updated.dd and were instructed to recover files by making use of file signatures. Instead of using a step-by-step process based on file system boundaries, we developed a Python script to take a disk image as an input, locate file signatures, properly recover user generated files without corruption, and generate a SHA-256 hash for each file recovered.

3 Methodology

We created a python script that takes a disk image as input, analyzes its contents, and compares and matches file signature information in order to recover files from the disk. We used the file signatures covered in class slides and also the GCK's File Signature Table. Depending on the file extension type, we provided the respective data, such as the header, footer, footer trailer, and file extension pertaining to the specific file. The file signatures are instantiated as a list at the beginning of our script. As our script reads a disk image, our main method iterates through the bytes in the disk image, searching for matches to the included file signatures. If the program finds a header, file count is incremented and a name for the current file is created according to the order in which it was recovered (File_1.extension, File_2.extension, etc.). The program then searches for a footer and footer trailer, where each is applicable. It saves the end of file offset to evaluate the file size. After this, a new file is created, bytes from the disk image are read into a variable and then that variable is written to the new file, according to the file size. This process is repeated until all files have been found and the end of the disk image is reached. As our program recovers these files, information about each file's starting and ending offsets, along with its name, file extension, and SHA-256 hash value is appended to an output string and relayed after file recovery completion. Our group was able to successfully recover the thirteen files held within the disk image.

Our group used several different references for this project. Reference [1] for creating a directory within our code, [2] for handling errors when user incorrectly executes program, [3] for granting read, write, and execution access from within our program, [4] to help determine the best data structure for our file extension signature information, [5] to add elapsed time feature to program output, [6] as a reference for file signature information, and [7] to discuss the .docx file contents and false-positive file recovery.

The files recovered were as follows:

- A .jpg file depicting Auburn University's logo
- A .avi file containing a video of a black bear
- A .pdf file containing A Tale of Two Cities by Charles Dickens
- A .png file with a transparent background with multicolored dice
- A .jpg file depicting many different national flags
- A .bmp file of purple flowers
- A .pdf file of Great Expectations by Charles Dickens
- A .jpg file containing a poster for the movie Iron Man
- A .gif file containing a blue-and-red psychedelic looped video
- A .gif file that contained a looping video of a Minion from the movie Despicable Me wearing red sirens on its head
- A .avi file depicting a filmed aerial view of the ocean
- A .docx file containing an image from the movie A Christmas Story
- A .mpg file containing a video of a simulation of the universe and outer-space

4 Results

Program output:

```
$ ./FileRecovery.py Project2Updated.dd
Disk image name:  Project2Updated.dd
```

This disk image contains 13 files.

File name: File_1.JPG
Start Offset: 0x38000
End Offset: 0x3b046
SHA-256: 7e8b819fbb740694afea232c0224043d4e66934ffa41b9ccb87de868a6046a1f

File name: File_2.AVI
Start Offset: 0x3c000
End Offset: 0x1be0a80
SHA-256: 1e424df16136eb568113dfeaec0142fedbdef76838d3c6b995ba4ce4a5a7df16

File name: File_3.PDF
Start Offset: 0x1be1000
End Offset: 0x1dd8482
SHA-256: f646a8d117664df7d3df7f0f28f308e6c72f24f77bb0b96fed13bf59b9f9a29d

File name: File_4.PNG
Start Offset: 0x1dd9000
End Offset: 0x1e10a6c
SHA-256: 1f07008c776f10b8386b2e55a2680a1afa56b02cfe348f3c6cab12f313fe79e9

File name: File_5.JPG
Start Offset: 0x1e11000
End Offset: 0x1e27983
SHA-256: f9548466e8ac2e3f3091aca2039fa5a668b396f56f379f68052ff58051f39a35

File name: File_6.BMP
Start Offset: 0x1e28000
End Offset: 0x1e3b076
SHA-256: e03847846808d152d5ecbc9e4477eee28d92e4930a5c0db4bffda4d9b7a27dfc

File name: File_7.PDF
Start Offset: 0x1e3c000
End Offset: 0x2147c6d
SHA-256: 673855d6fd075236d6d6154624012a8182ecaf2cb632f621dfb9ae46da3f8f03

File name: File_8.JPG
Start Offset: 0x2148000
End Offset: 0x214d720
SHA-256: 6e826ef8018bb3321be753e7bfe4a4c183f18fb52bf0b500e9890fc5f780ae37

File name: File_9.GIF
Start Offset: 0x214e000
End Offset: 0x23d59d6
SHA-256: 5adfdff554fc0e507acd98414b5b8346546bd8b8233afd3375fc980d023cdbe7

File name: File_10.GIF
Start Offset: 0x23d6000
End Offset: 0x24261df
SHA-256: 54208f131e4d0be296b83e24092bb6538aa104ff23604641b7210fb3374d0e6c

File name: File_11.AVI
Start Offset: 0x2427000
End Offset: 0x2d9a364
SHA-256: 145d0a0e4870e02b0d80432c4b945add0c8b5178705a8ef21816d84a6ecd8aa6

File name: File_12.DOCX
Start Offset: 0x2d9b000
End Offset: 0x2dbc5a8
SHA-256: e45ca13b827480aee1814b3bcb18b2dbf4ed96207a5053573da25494c7c844f6

```
File name: File_13.MPG
Start Offset: 0x2dbd000
End Offset: 0x2faf89d
SHA-256: 6bf16deca3acb66d65aa00d9275df036c1c93732f6f2e66f9799937b962c73f4

Recovered files are located in ~/RecoveredFiles

Time elapsed:
16.335257530212402
```

The output of the program prints the file names similar to the sample provided by the instructor, the starting and ending offsets, and the SHA-256 of each file. A we can see from the above, the program works and provides output as expected.

5 Conclusions

Our team found that using a Python script was a good way to extract files from a disk image without corrupting the disk. We were able to collect all 13 files for examination using our code. We did have an issue with getting false-positives on .BMP files due to their short headers, and it is crucial to check files with such headers to ensure that they are valid files.

6 References

[1] https://www.geeksforgeeks.org/python-os-mkdir-method/
[2] https://www.geeksforgeeks.org/python-check-if-a-file-or-directory-exists-2/
[3] https://www.tutorialspoint.com/python/os_chmod.htm
[4] https://github.com/schlerp/pyfsig
[5] https://stackoverflow.com/questions/7370801/how-to-measure-elapsed-time-in-python
[6] https://www.garykessler.net/library/file_sigs.htm
[7] Professor Jason Cuneo