Project 2

Aaron Smith

December 3, 2021

Executive Summary

The purpose of this project is to recover files from a disk image where the file system is not familiar to the forensic analyst. The only thing we know about the disk is the files are stores in a consecutive piece of memory.

First step in the process is to determine what type of files are on the disk image. File types can be determined using file signatures, which are unique identifiers that are found at the beginning, headers, and sometimes the end, footers, of the files. A python script “FileRecovery.py” was developed to go through the contents of the disk using both header and footer information to determine file size, then it recovers the files.

Some file types have irregularities such as not having footers or having multiple footers that cause false positives. For example, PDF file footers can be found throughout the entire file length, so there are different methods for carving these.

By using these methods, a total of fourteen files were carved from the provided disk image.

Table of Contents

Table of Contents

1 Introduction

A disk image was recovered that contained files stored in a consecutive set of memory. The file system was unknown, but because they were stored back-to-back, they can still be recovered using a script.

2 Methodology

The python script that was developed knowing that file types have specific headers and footers that are attached to them. If we know the file headers and footers, the file sizes can be determined to then be extracted.

2.1 File Headers and Footers

File headers and footers are public data available from many different sources, but the source used here was a library consisting of most mainly used file types. Figure 2.1.1 shows the files that were required to be inspected along with their respective headers and footers.

Figure 2.1.1

File Type	Header Signature	Footer Signature
MPG	\x00\x00\x01\xB3\x00	\x00\x00\x00\x01\xB7
PDF	\x25\x50\x44\x46	\x0A\x25\x25\x45\x4F\x46\x0A
PDF	\x25\x50\x44\x46	\x0D\x0A\x25\x25\x45\x4F\x46\x0D\x0A
PDF	\x25\x50\x44\x46	\x0A\x25\x25\x45\x4F\x46
PDF	\x25\x50\x44\x46	\x0D\x25\x25\x45\x4F\x46\x0D
BMP	\x42\x4D....\x00\x00\x00\x00	NONE

GIF	\x47\x49\x46\x38\x37\x61	\x00\x00\x3B
GIF	\x47\x49\x46\x38\x39\x61	\x00\x00\x3B
JPG	\xFF\xD8\xFF\xE0	\xFF\xD9
JPG	\xFF\xD8\xFF\xE1	\xFF\xD9
JPG	\xFF\xD8\xFF\xE2	\xFF\xD9
JPG	\xFF\xD8\xFF\xE8	\xFF\xD9
JPG	\xFF\xD8\xFF\xD8	\xFF\xD9
DOCX	\x50\x4B\x03\x04\x14\x00\x06\x00	\x50\x4B\x05\x06
AVI	\x52\x49\x46\x46....\x41\x56\x49\x20\x4C\x49\x53\x54	NONE
PNG	\x89\x50\x4E\x47\x0D\x0A\x1A\x0A	\x49\x45\x4E\x44\xAE\x42\x60\x82

The headers and footers were put into a list into the Python script, so they could be easily identified. A header was identified to log the starting point, then a footer was identified to log the end point and thus logging the whole file size. Knowing the beginning and end points of a file made it easier to carve out for recovery.

2.2 Hashing

After the file was recovered, the entire file was sent through a SHA-256 hashing algorithm to give us a SHA-256. Hashing a recovered file is important because it can help determine if the file has been tampered with afterwards.

2.3 Unique File Types

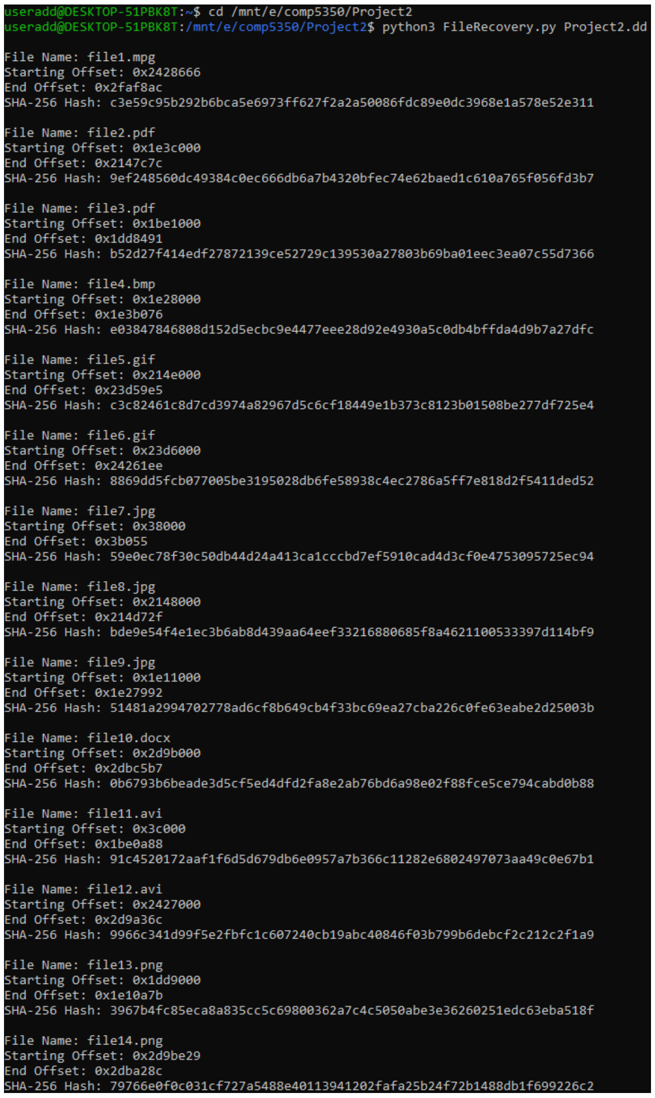
Some file types are unique when it comes to their file signatures, so it is important to workaround these differences. Some cases include bytes that are the same as a file signature located elsewhere in the data, and multiple instances of footers being found.

In the case where signature bytes are located within the data itself instead of in the header or footer, we need to look at other attributes of that file type to help determine whether the located signature is a false positive or not. An example of this is BMP files. BMP files have reserved bytes after the start of the file that are filled with zero values, and if these bytes are identified, we could then determine if the data was a false positive or not.

The other case happened namely with PDF files. PDF files are unique because they have multiple file footers located throughout the entirety of its bytes, so it’s important to locate the very last footer so that the whole thing can be carved. The best way to implement this is to identify the header of the file that could come after the PDF file to determine which of the footers really is the last one.

2.4 Screenshot

The following screenshot is used to provide the location of the files using starting and ending offsets and therefore their size. Their SHA-256 hashes are also provided.



```
useradd@DESKTOP-51PBK8T:~$ cd /mnt/e/comp5350/Project2
useradd@DESKTOP-51PBK8T:/mnt/e/comp5350/Project2$ python3 FileRecovery.py Project2.dd

File Name: file1.mpg
Starting Offset: 0x2428666
End Offset: 0x2faf8ac
SHA-256 Hash: c3e59c95b292b6bca5e6973ff627f2a2a50086fdc89e0dc3968e1a578e52e311

File Name: file2.pdf
Starting Offset: 0x1e3c000
End Offset: 0x2147c7c
SHA-256 Hash: 9ef248560dc49384c0ec666db6a7b4320bfec74e62baed1c610a765f056fd3b7

File Name: file3.pdf
Starting Offset: 0x1be1000
End Offset: 0x1dd8491
SHA-256 Hash: b52d27f414edf27872139ce52729c139530a27803b69ba01eec3ea07c55d7366

File Name: file4.bmp
Starting Offset: 0x1e28000
End Offset: 0x1e3b076
SHA-256 Hash: e03847846808d152d5ecbc9e4477eee28d92e4930a5c0db4bffda4d9b7a27dfc

File Name: file5.gif
Starting Offset: 0x214e000
End Offset: 0x23d59e5
SHA-256 Hash: c3c82461c8d7cd3974a82967d5c6cf18449e1b373c8123b01508be277df725e4

File Name: file6.gif
Starting Offset: 0x23d6000
End Offset: 0x24261ee
SHA-256 Hash: 8869dd5fcb077005be3195028db6fe58938c4ec2786a5ff7e818d2f5411ded52

File Name: file7.jpg
Starting Offset: 0x38000
End Offset: 0x3b055
SHA-256 Hash: 59e0ec78f30c50db44d24a413ca1cccbd7ef5910cad4d3cf0e4753095725ec94

File Name: file8.jpg
Starting Offset: 0x2148000
End Offset: 0x214d72f
SHA-256 Hash: bde9e54f4e1ec3b6ab8d439aa64eef33216880685f8a4621100533397d114bf9

File Name: file9.jpg
Starting Offset: 0x1e11000
End Offset: 0x1e27992
SHA-256 Hash: 51481a2994702778ad6cf8b649cb4f33bc69ea27cba226c0fe63eabe2d25003b

File Name: file10.docx
Starting Offset: 0x2d9b000
End Offset: 0x2dbc5b7
SHA-256 Hash: 0b6793b6beade3d5cf5ed4dfd2fa8e2ab76bd6a98e02f88fce5ce794cabd0b88

File Name: file11.avi
Starting Offset: 0x3c000
End Offset: 0x1be0a88
SHA-256 Hash: 91c4520172aaf1f6d5d679db6e0957a7b366c11282e6802497073aa49c0e67b1

File Name: file12.avi
Starting Offset: 0x2427000
End Offset: 0x2d9a36c
SHA-256 Hash: 9966c341d99f5e2fbfc1c607240cb19abc40846f03b799b6debcf2c212c2f1a9

File Name: file13.png
Starting Offset: 0x1dd9000
End Offset: 0x1e10a7b
SHA-256 Hash: 3967b4fc85eca8a835cc5c69800362a7c4c5050abe3e36260251edc63eba518f

File Name: file14.png
Starting Offset: 0x2d9be29
End Offset: 0x2dba28c
SHA-256 Hash: 79766e0f0c031cf727a5488e40113941202fafa25b24f72b1488db1f699226c2
```

3 Conclusion

A total of fourteen files were recovered using the developed script. They are attached with this.