Project 2 Report

Team Member Name(s):

Carlton Schmieder

Sean O’Connor

Myles Parkhurst

Date:

12/3/2021

Table of Contents

Table of Contents

1 Executive Summary

The team carried out a digital forensics investigation on a disk image with unknown contents. Tasked with creating a python script to automate the recovery process, the team combined knowledge of the desired file headers and trailers found at https://www.garykessler.net/library/file_sigs.html with previous experience in python and forensics investigations to recover fourteen files. The contents of the files as well as their implications will be expanded upon in the rest of the report.

2 Problem Description

For this project, we were tasked with developing a Python script that will that a disk image as an input. Once it reads in the disk image file, the script will then locate file signatures and properly recover user generated files without corruption. The script then needs to take the file and generate a SHA-256 hash for each file recovered. After all the information is gathered, the script then needs to print the number of files recovered as well as the start and end offsets. Lastly, the script needs to print where the files are being placed on the local machine.

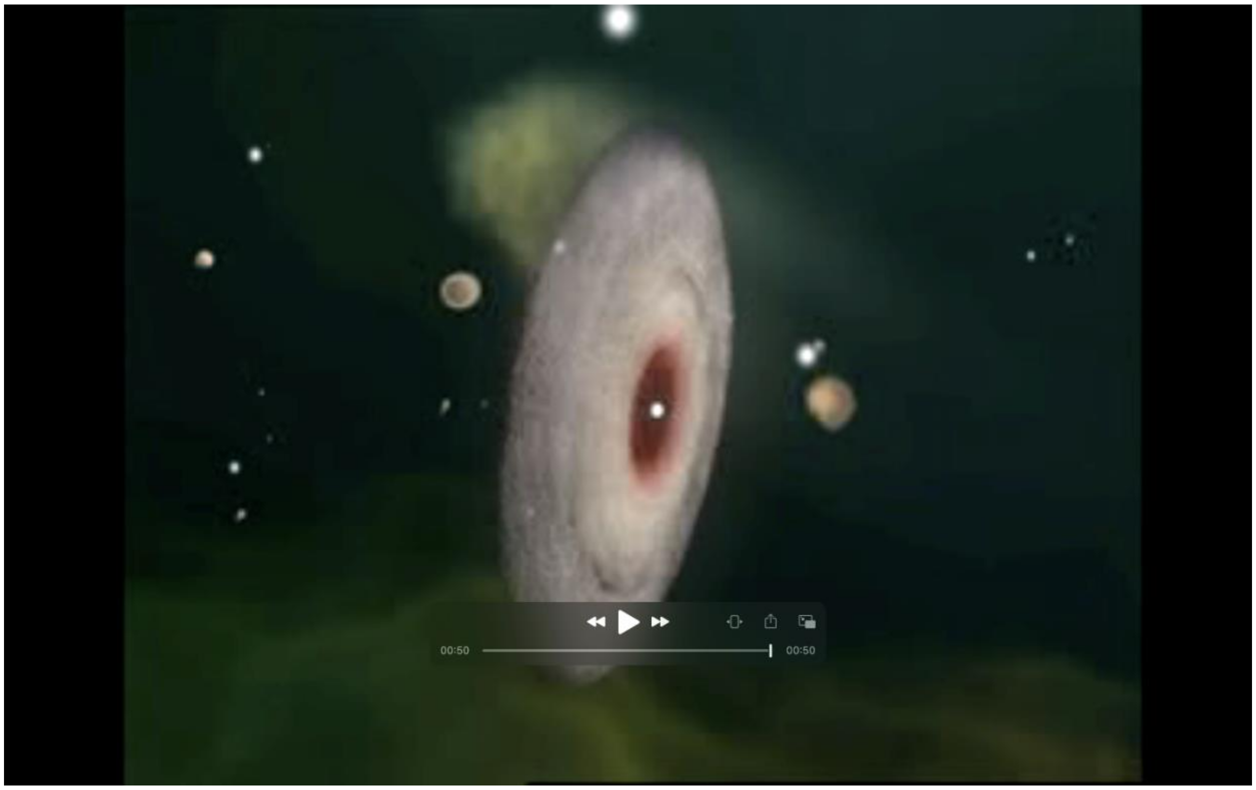
3 Description of Analysis Techniques Utilized

In order to achieve our desired results and fulfill the entirety of the problem statement, several import statements were used. These import statements were re, hashlib, struct, sys, and binascii. The re import was used to convert our headers into searchable elements. Hashlib was used to produce a sha256 string from the filename. Struct was utilized to help handle binary formatting. To allow the script to accept parameters from terminal, the script implemented the sys import, and used binascii to help with conversion. The team created a list of lists to hold all of the possible outcomes in terms of how a file could be found. To achieve this, the first element in a list element was the file extension (as a string) followed by the desired header and a footer (if applicable). If the file extension did not require a footer, a “None” element was placed instead to indicate this difference. With this approach, the team was able to account for each different file header, footer, etc. For example, a .jpg file had 5 different combinations of headers that could potentially be found; therefore, there are five entries into the list that contain each respective possibility. In addition, objects and variables such as several Boolean expressions and arrays were used to help ensure no duplicate files were produced, and that no offsets would overlap each other. The entirety of the script is contained inside a for loop that iterates through the file extension list and searches for each one in the disk image. Essentially, the disk image is being searched eighteen times to account for each different possibility of file existence. Once the disk drive has been read in and assigned to a variable, the re import utilizes the regex command “compile” and “finditer” to locate every possibility of a file header for the particular file extension. Once this has been completed, the script can traverse through each iteration of a file header found to search and carve out a found file. Once it reaches this point, it assigns the offset to a variable so that it can be stored as the starting offset and checks to make sure the header has not been utilized in a previous iteration of the search. From there, it checks to see if the file entry contains a footer or if the file size is determined in the header. If the file size is in the header, it grabs that information and converts from little endian hex into a long value (this is where the binascii import is used). With this long value and the starting offset, the file can be carved out. Afterwards, the ending offset is found dependent on which type of file extension is being dealt with (bmp or avi in this case). If the file header has a file footer, the script, in a way similar to the header technique used, converts the footer into a searchable element using the re import. Once again, it is checked that the footer is unique and has not been used for a prior iteration. The footer is searched for and once found, its proper offset is stored in a variable. Afterwards, the script checks to make sure the file is not a pdf or docx. If it is a docx file, the end offset adds 18 to the current value. If the file has a pdf extension the script runs through a set of commands to ensure the found footer is the proper end of file tag for the current header. After this, script has the proper starting and ending offset. The script took these values and grabbed the data between the offsets and assigned it to its own variable. Then the script creates a file name and opens a writer which then wrote the file contents to the unique file name. Once complete, the file writer was closed and now the complete file carved from the disk drive is created. The proper variables are printed out to the console to show the user that the file had been carved out and to illustrate where exactly the file was carved from based on the starting and ending offsets printed out in their hex format. Throughout the running of this script, the Boolean expressions are used to signify where exactly the code should execute next. Once one of the 18 file entries has been searched for in the disk drive, the script grabs the next entry and restarts the process until completion. Completion is signified when each entry has properly been searched for in the disk drive. Using this process, the team was able to find fourteen total files in the disk drive which are described under the screenshots section of this report.

4 Screenshots

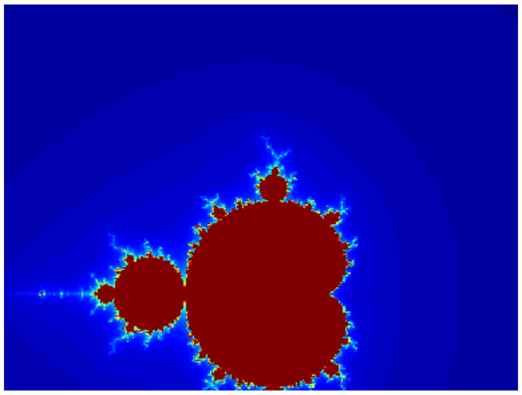
FILE 1

File 1 is a MPG file that contained a video zoom through a galaxy



FILE 2

File 2 is a GIF file that zooms in and branches the more it zooms in



FILE 3

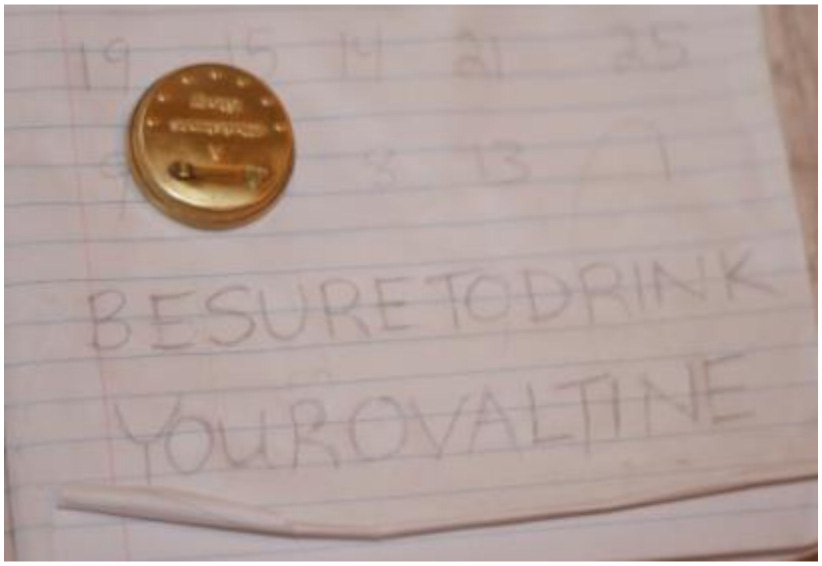
File 3 is a GIF file that contains a minion with 3 flashing lights on its head (From Despicable Me 2)

FILE 4

File 4 is PNG file that contains a picture of 4 differently colored dice

FILE 5

File 5 is a PNG file that contains a picture with a pin and the words “BESURETODRINK YOUROVALTINE”



FILE 6

File 6 is a JPG file that contains a picture of the Auburn Logo. (War Eagle!)

FILE 7

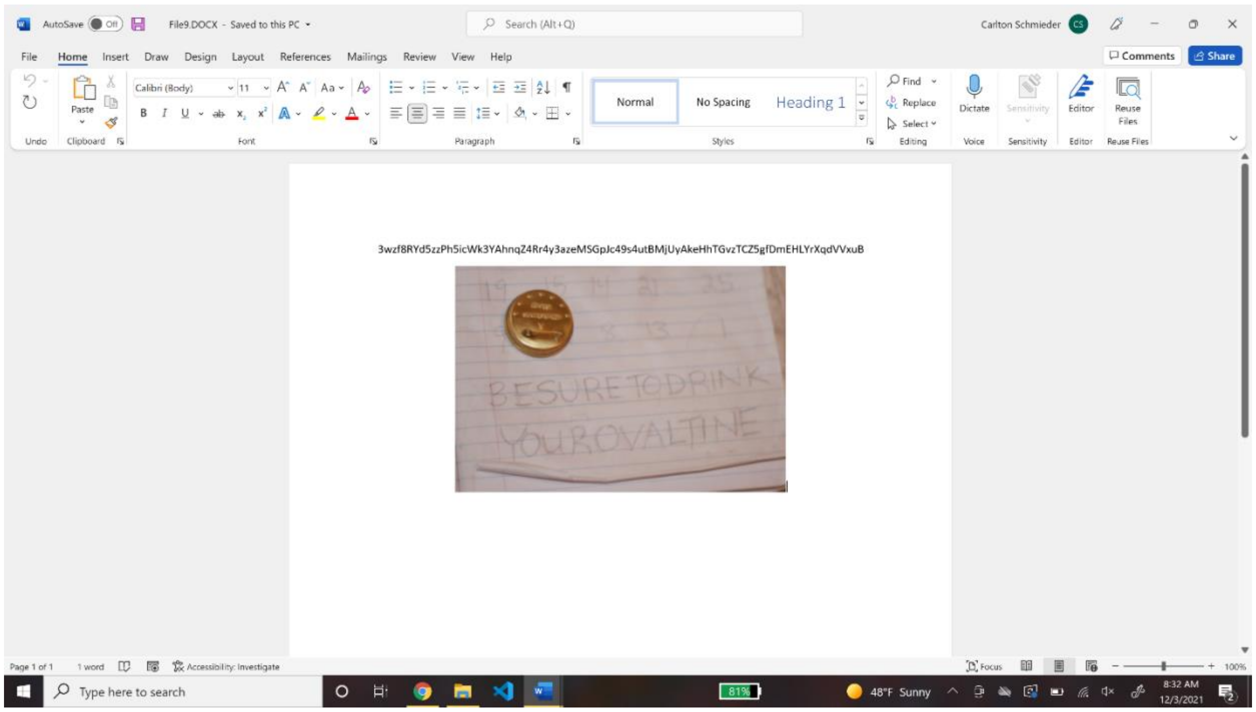
File 7 is a JPG file that contains a picture of the movie poster for Iron Man

FILE 8

File 8 is a JPG file that contains a picture of the flags around the world.

FILE 9

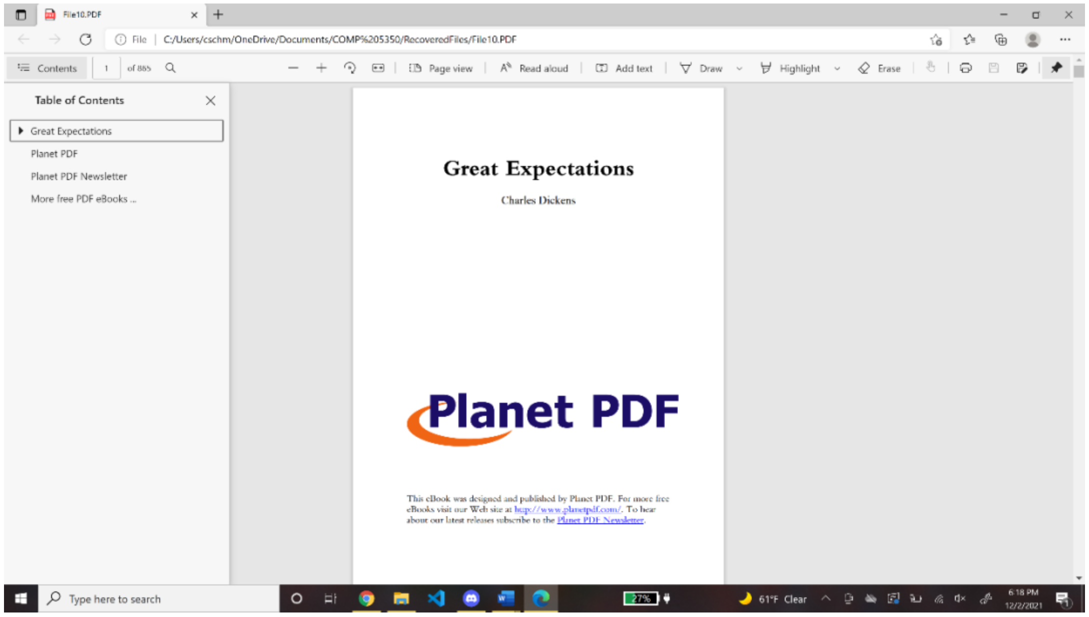
File 9 is a DOCX file that contains a picture



3wzf8RYd5zzPh5icWk3YAhnqZ4Rr4y3azeMSGpJc49s4utBMjUyAkeHhTGvzTCZ5gfDmEHLYrXqdVVxuB

FILE 10

File 10 is a PDF that contains Great Expectations. All 865 pages included.



Great Expectations

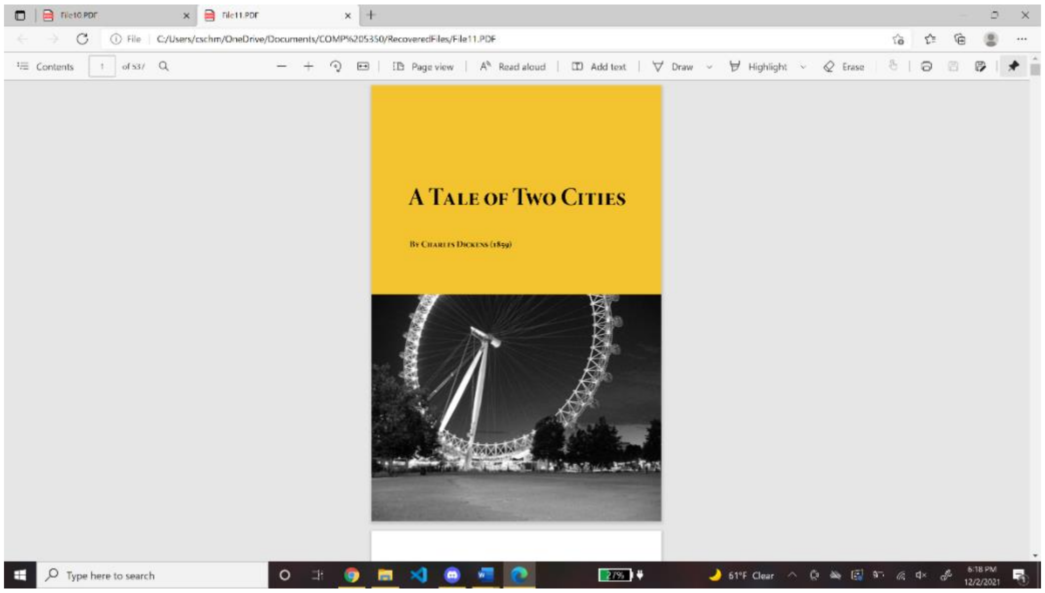
Charles Dickens

Planet PDF

This eBook was designed and published by Planet PDF. For more free eBooks visit our Web site at http://www.planetpdf.com/. To hear about our latest releases subscribe to the Planet PDF Newsletter.

FILE 11

File 11 is a PDF file that contains A Tale of Two Cities. All 537 pages included.



A Tale of Two Cities

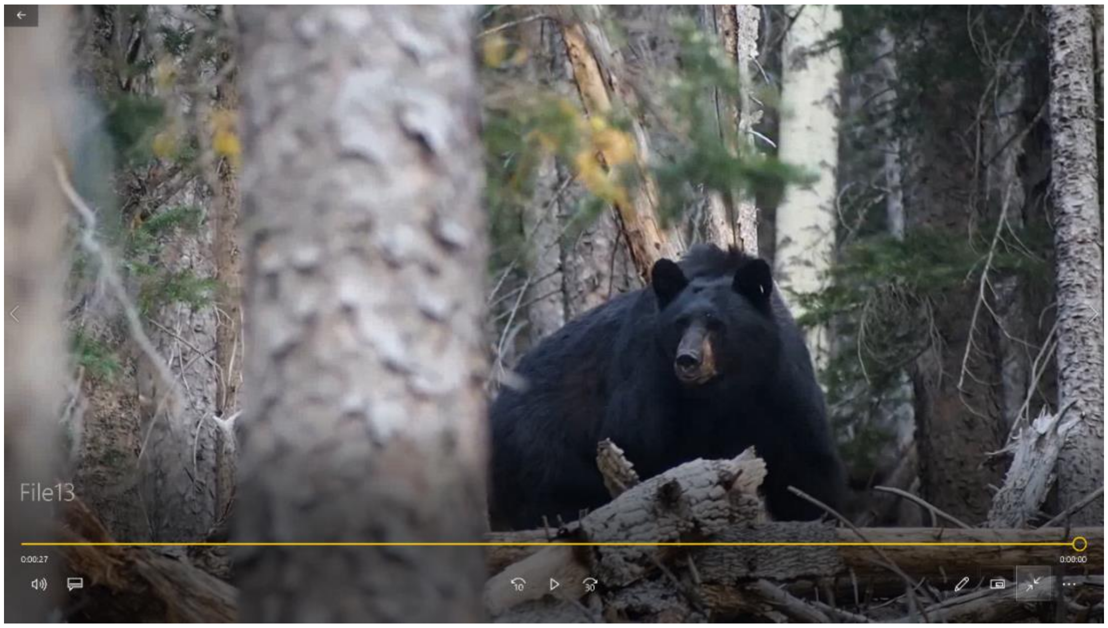
By Charles Dickens (1859)

FILE 12

File 12 is a BMP file that contains a picture of purple flowers (Believed to be China Asters)

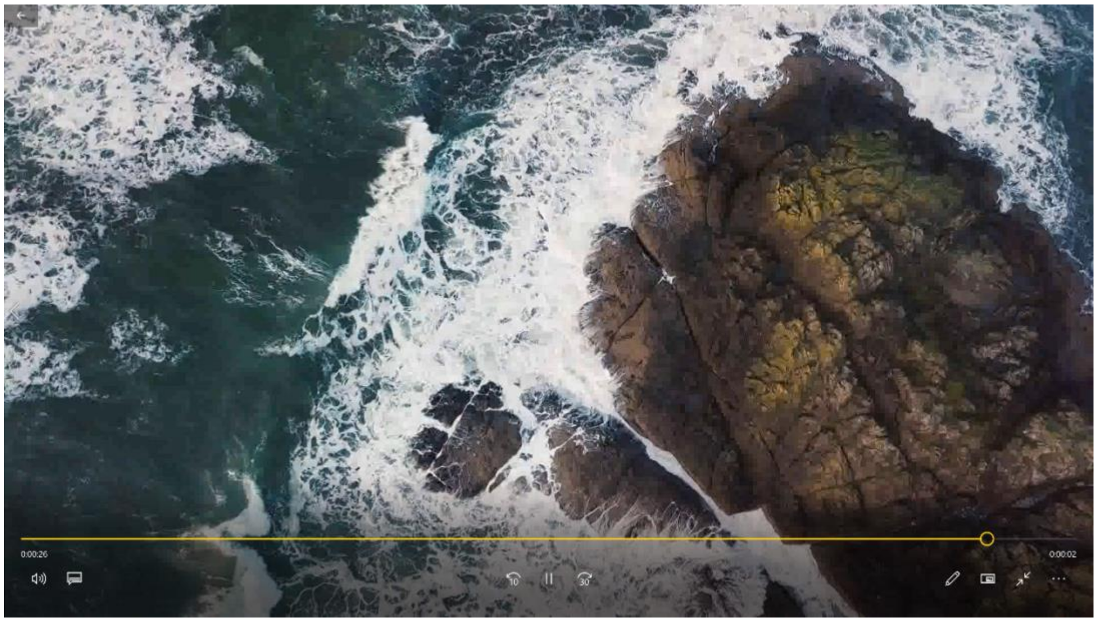
FILE 13

File 13 is a AVI file that contains a 28 second video of a massive black bear.



FILE 14

File 14 is a AVI file that contains a 28 second video of waves crashing on rocks.



FILE TABLE

File Name: **File1.MPG**

Starting Offset: 0x2dbd000

End Offset: 0x2faf8ac

SHA-256 Hash: c9ed8592d0b31b24e5a7286469497cd817e7c32dd6a9347891db8c27c26d0153

File Name: **File2.GIF**

Starting Offset: 0x214e000

End Offset: 0x23d59e5

SHA-256 Hash: c3c82461c8d7cd3974a82967d5c6cf18449e1b373c8123b01508be277df725e4

File Name: **File3.GIF**

Starting Offset: 0x23d6000

End Offset: 0x24261ee

SHA-256 Hash: 8869dd5fcb077005be3195028db6fe58938c4ec2786a5ff7e818d2f5411ded52

File Name: **File4.PNG**

Starting Offset: 0x1dd9000

End Offset: 0x1e10a7b

SHA-256 Hash: 3967b4fc85eca8a835cc5c69800362a7c4c5050abe3e36260251edc63eba518f

File Name: **File5.PNG**

Starting Offset: 0x2d9be29

End Offset: 0x2dba28c

SHA-256 Hash: 79766e0f0c031cf727a5488e40113941202fafa25b24f72b1488db1f699226c2

File Name: **File6.JPG**

Starting Offset: 0x38000

End Offset: 0x3b055

SHA-256 Hash: 59e0ec78f30c50db44d24a413ca1cccbd7ef5910cad4d3cf0e4753095725ec94

File Name: **File7.JPG**

Starting Offset: 0x2148000

End Offset: 0x214d72f

SHA-256 Hash: bde9e54f4e1ec3b6ab8d439aa64eef33216880685f8a4621100533397d114bf9

File Name: **File8.JPG**

Starting Offset: 0x1e11000

End Offset: 0x1e27992

SHA-256 Hash: 51481a2994702778ad6cf8b649cb4f33bc69ea27cba226c0fe63eabe2d25003b

File Name: **File9.DOCX**

Starting Offset: 0x2d9b000

End Offset: 0x2dbc5b7

SHA-256 Hash: 0b6793b6beade3d5cf5ed4dfd2fa8e2ab76bd6a98e02f88fce5ce794cabd0b88

File Name: **File10.PDF**

Starting Offset: 0x1e3c000

End Offset: 0x2147c7c

SHA-256 Hash: 9ef248560dc49384c0ec666db6a7b4320bfec74e62baed1c610a765f056fd3b7

File Name: **File11.PDF**

Starting Offset: 0x1be1000

End Offset: 0x1dd8491

SHA-256 Hash: b52d27f414edf27872139ce52729c139530a27803b69ba01eec3ea07c55d7366

File Name: **File12.BMP**

Starting Offset: 0x1e28000

End Offset: 0x1e3b076

SHA-256 Hash: e03847846808d152d5ecbc9e4477eee28d92e4930a5c0db4bffda4d9b7a27dfc

File Name: **File13.AVI**

Starting Offset: 0x3c000

End Offset: 0x1be0a88

SHA-256 Hash: 91c4520172aaf1f6d5d679db6e0957a7b366c11282e6802497073aa49c0e67b1

File Name: **File14.AVI**

Starting Offset: 0x2427000

End Offset: 0x2d9a36c

SHA-256 Hash: 9966c341d99f5e2fbfc1c607240cb19abc40846f03b799b6debcf2c212c2f1a9

5 Conclusions and Recommendations

Conclusion and Recommendation:

In total there were fourteen total files recovered. That includes 1 MPG, 2 GIFs, 2PNGs, 3 JPGs, 1 DOCX, 2 PDFs, 1 BMP, and 2 AVIs. The contents of the files varied wildly from mathematical models and movies posters to bear videos and flag charts. Investigators utilized a python script to expedite the file recovery process. File signatures and trailers were used to locate and later extract files from the disk image automatically. The script allowed for a more streamlined approach while giving investigators more confidence in the results due to the distancing from possible human error. There does not appear to be any real connection between the files which leaves the investigation little direction without the addition of further evidence.