AUBURN
UNIVERSITY

Project 2

Group #7

Blake Moore and Nico Marthe

11/3/2023

1 Executive Summary

We were tasked with creating a python script that could locate and recover files hidden on a disk-drive by recognizing their file signatures. This would allow for an easy and automatic process for recovering files during a forensic investigation. The goal of the script was to take in a disk image, search the disk image for any file signatures and then recover them from the disk image using their starting and ending offsets. The script would then generate a SHA-256 hash for each file found and recovered from the disk image, then display those as the output of the script. This script is configured to recover MPG, PDF, BMP, GIF, JPG, DOCX, AVI, and PNG files. Using this, we recovered and hashed 14 different files of the various types listed. To see the summary results, see Figures 1 and 2. The following report covers our methodology for how the script works as well as the results from testing of the script.

2 Collaboration Summary

This project was worked on by both Blake Moore and Nico Marthe. We split our creation of the script based on the different file types required for the script to read and recover. Working together we were able to write the python script to accept each required file type and then collaborated in the writing process of this report.

Table of Contents

1 Executive Summary 2

2 Collaboration Summary 3

List of Figures 5

3 Problem Description 6

4 Analysis Techniques 6

5 Tables and Screenshots 7

6 Conclusions and Recommendations 20

List of Figures

Figure 1: Code execution and partial output……………………………………………………………………..8

Figure 2: All the recovered files in the working directory.…………………………………………………………8

Figure 3: Main method of python script that iterates through each 512 byte section of the disk drive searching for files……9

Figure 4: MPG file recovery, which is the generic recovery process.……………………………………………10

Figure 5: BMP file recovery, which utilizes the "dd" command.…………………………………………………11

Figure 6: Properly recovered File1.jpg…………………………………………………………………………...11

Figure 7: Properly recovered File2.avi…………………………………………………………………………...12

Figure 8: Properly recovered File3.pdf…………………………………………………………………………...12

Figure 9: Properly recovered File4.png…………………………………………………………………………..13

Figure 10: Properly recovered File5.jpg…………………………………………………………………………13

Figure 11: Properly recovered File6.bmp………………………………………………………………………..14

Figure 12: Properly recovered File7.pdf…………………………………………………………………………14

Figure 13: Properly recovered File8.jpg…………………………………………………………………………15

Figure 14: Properly recovered File9.gif…………………………………………………………………………15

Figure 15: Properly recovered File10.gif………………………………………………………………………..16

Figure 16: Properly recovered File11.avi………………………………………………………………………..16

Figure 17: Properly recovered File12.docx……………………………………………………………………...17

Figure 18: Properly recovered File13.png……………………………………………………………………….17

Figure 19: Properly recovered File14.mpg………………………………………………………………………18

Figure 20: TA email detailing issues with .zip files ……………………………………………………………..18

Figure 21: Output to user, including start/end offsets and hashes………………………………………………19

Figure 22: Continued output to user, including start/end offsets and hashes……………………………………19

3 Problem Description

The initial task was to write a python script that could recover the files hidden on a disk image using file signatures. The main problem that came about with this was understanding how to determine file signatures for each file type as well as properly implementing a python script to recover the files, as neither of us has ever had to write code like this before. With a lot of trial and error we managed to write a script that works for each of the necessary file types and generates the SHA-256 results as requested.

4 Analysis Techniques

In order to analyze the disk drive provided we wrote a python script to automatically find, recover, and hash any files found on the disk drive. To execute the script properly, the user passes the disk drive in as a command line argument. To see this, see Figure 1.

The code reads in this disk drive and examines it 512 bytes at a time. For each section of 512 bytes, the code searches for each of the different types of file signatures near the beginning of the section. If it finds a file, it recovers and hashes it, then outputs the results to user. After that process, or if it does not find a file for that section, it iterates to the next 512 bytes and repeats the process till it reaches the end of the disk drive. To see this code, see Figure 3.

The different recovery processes for each of the file types were very similar to one another. The code would search the beginning of the current section of 512 bytes for its respective file header. We used the file signatures found in the link provided in the assignment to know which headers to search for with each file type. If a file header is found, the code enters a recovery mode, and calculates the offset where the file begins. After this, it searches for the file trailers, which we found at the same link as the file headers. If a file trailer is found, the file ending offset is calculated. Using these two offsets, the file found is written to a new file of the appropriate file extension. The code then generates a SHA-256 hash of the file, and outputs all the information about the file to the user. To see the generic recovery process, see Figure 4, which shows the MPG file recovery method.

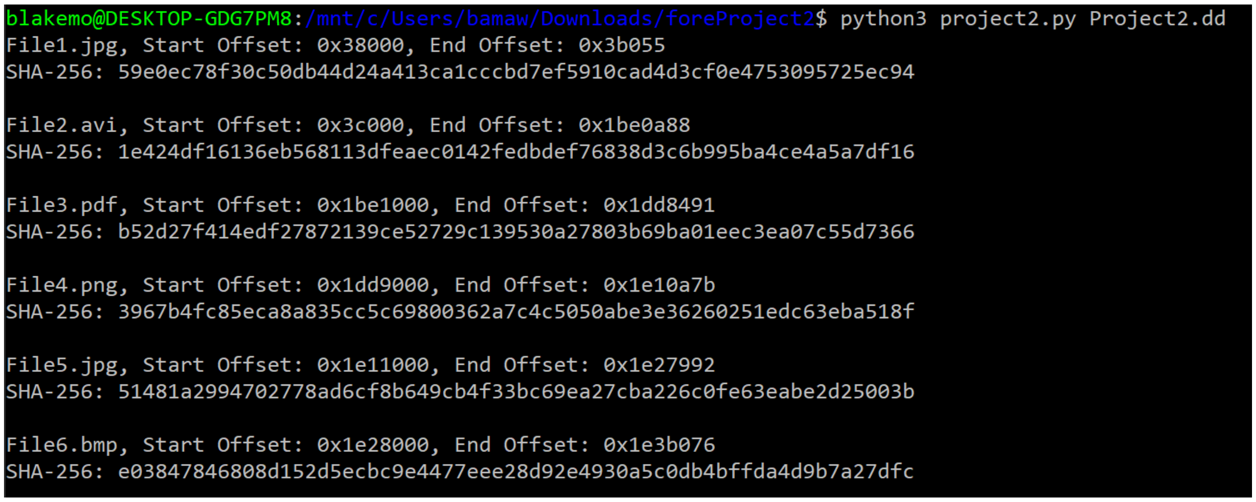
Each file type is different, so the recovery method does vary slightly at times. For example, .bmp files do not have file trailers. However, the file length can be found after the file header, meaning we could recover the file by sending a “dd” command to the operating system with the start offset and the file length. To see this, see Figure 5. There are a few other special circumstances with some of the file types, to see each of these in depth watch the video submitted alongside this paper in the Canvas submission.

Through the execution of the code, we recover 14 different files. To see proof of each of the files being recovered properly, see Figures 6 – 19. The only file type not recovered is .zip files. This is due to an issue with the disk drive provided, as we were told by the TA. To see proof of this, see Figure 20. We created some initial code to recover .zip files. To see more information about this, watch the video submitted alongside this paper in the Canvas submission.

In order to execute the script properly it must be executed in a linux shell, using python version 3.10 or newer. These constraints are due to sending “dd” commands to the operating system, and using a hashlib function that is only included in newer versions of python. To see the output to the user, the file offsets, and the hashes generated for each file, see Figures 21 and 22.

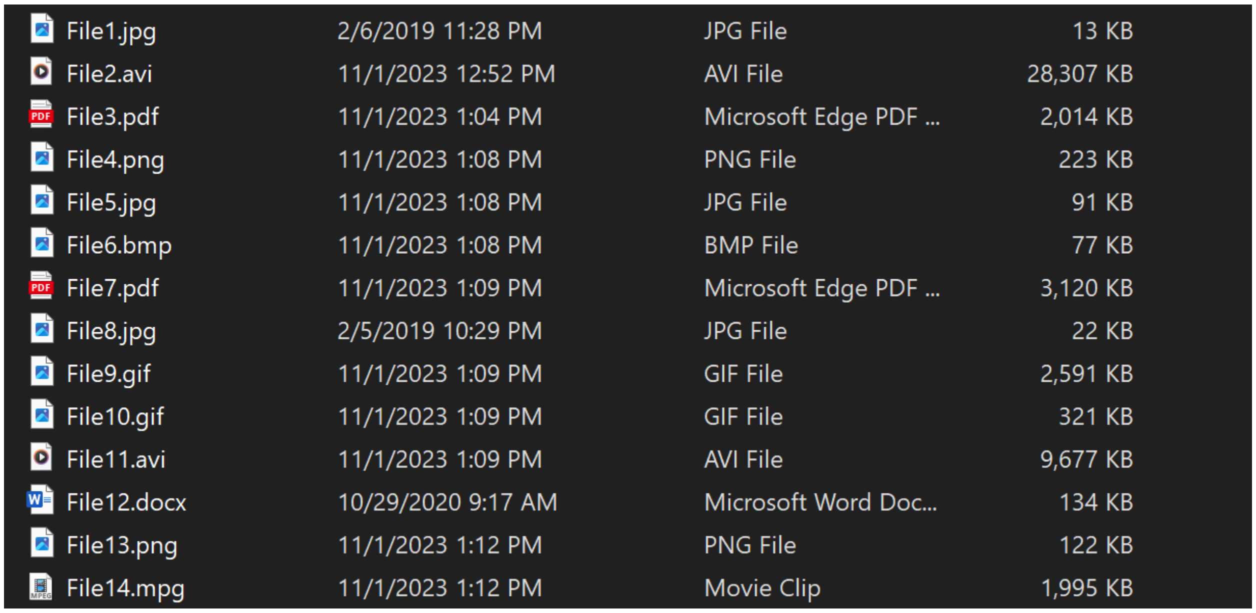
5 Tables and Screenshots

All screenshots referenced are shown here.



```
blakemo@DESKTOP-GDG7PM8:/mnt/c/Users/bamaw/Downloads/foreProject2$ python3 project2.py Project2.dd
File1.jpg, Start Offset: 0x38000, End Offset: 0x3b055
SHA-256: 59e0ec78f30c50db44d24a413ca1cccbd7ef5910cad4d3cf0e4753095725ec94

File2.avi, Start Offset: 0x3c000, End Offset: 0x1be0a88
SHA-256: 1e424df16136eb568113dfeaec0142fedbdef76838d3c6b995ba4ce4a5a7df16

File3.pdf, Start Offset: 0x1be1000, End Offset: 0x1dd8491
SHA-256: b52d27f414edf27872139ce52729c139530a27803b69ba01eec3ea07c55d7366

File4.png, Start Offset: 0x1dd9000, End Offset: 0x1e10a7b
SHA-256: 3967b4fc85eca8a835cc5c69800362a7c4c5050abe3e36260251edc63eba518f

File5.jpg, Start Offset: 0x1e11000, End Offset: 0x1e27992
SHA-256: 51481a2994702778ad6cf8b649cb4f33bc69ea27cba226c0fe63eabe2d25003b

File6.bmp, Start Offset: 0x1e28000, End Offset: 0x1e3b076
SHA-256: e03847846808d152d5ecbc9e4477eee28d92e4930a5c0db4bffda4d9b7a27dfc
```

Figure 1: Code execution and partial output



File1.jpg	2/6/2019 11:28 PM	JPG File	13 KB
File2.avi	11/1/2023 12:52 PM	AVI File	28,307 KB
File3.pdf	11/1/2023 1:04 PM	Microsoft Edge PDF ...	2,014 KB
File4.png	11/1/2023 1:08 PM	PNG File	223 KB
File5.jpg	11/1/2023 1:08 PM	JPG File	91 KB
File6.bmp	11/1/2023 1:08 PM	BMP File	77 KB
File7.pdf	11/1/2023 1:09 PM	Microsoft Edge PDF ...	3,120 KB
File8.jpg	2/5/2019 10:29 PM	JPG File	22 KB
File9.gif	11/1/2023 1:09 PM	GIF File	2,591 KB
File10.gif	11/1/2023 1:09 PM	GIF File	321 KB
File11.avi	11/1/2023 1:09 PM	AVI File	9,677 KB
File12.docx	10/29/2020 9:17 AM	Microsoft Word Doc...	134 KB
File13.png	11/1/2023 1:12 PM	PNG File	122 KB
File14.mpg	11/1/2023 1:12 PM	Movie Clip	1,995 KB

Figure 2: All the recovered files in the working directory.

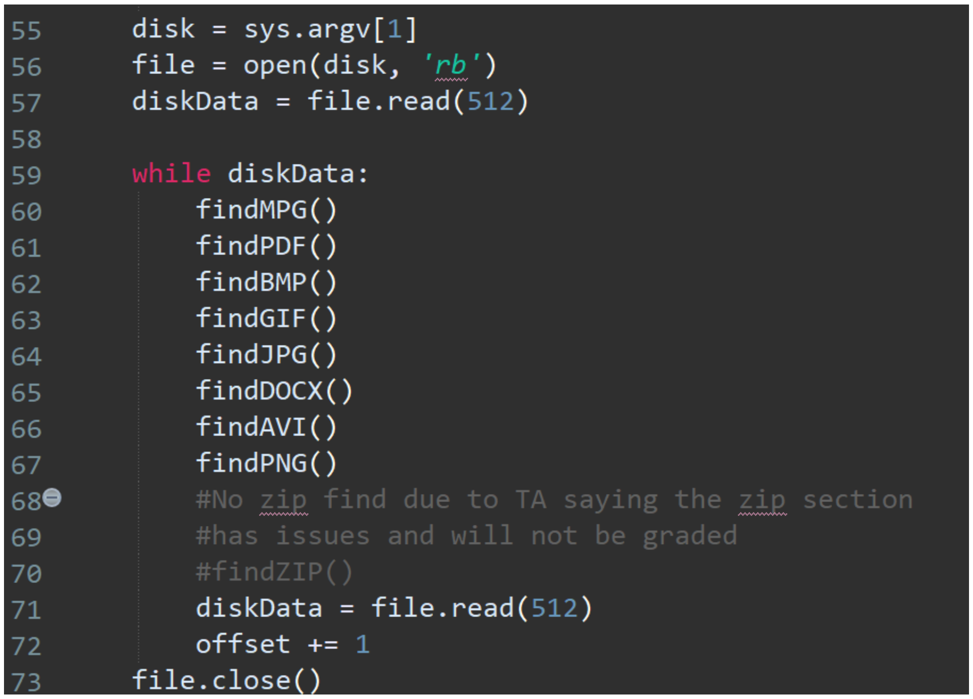
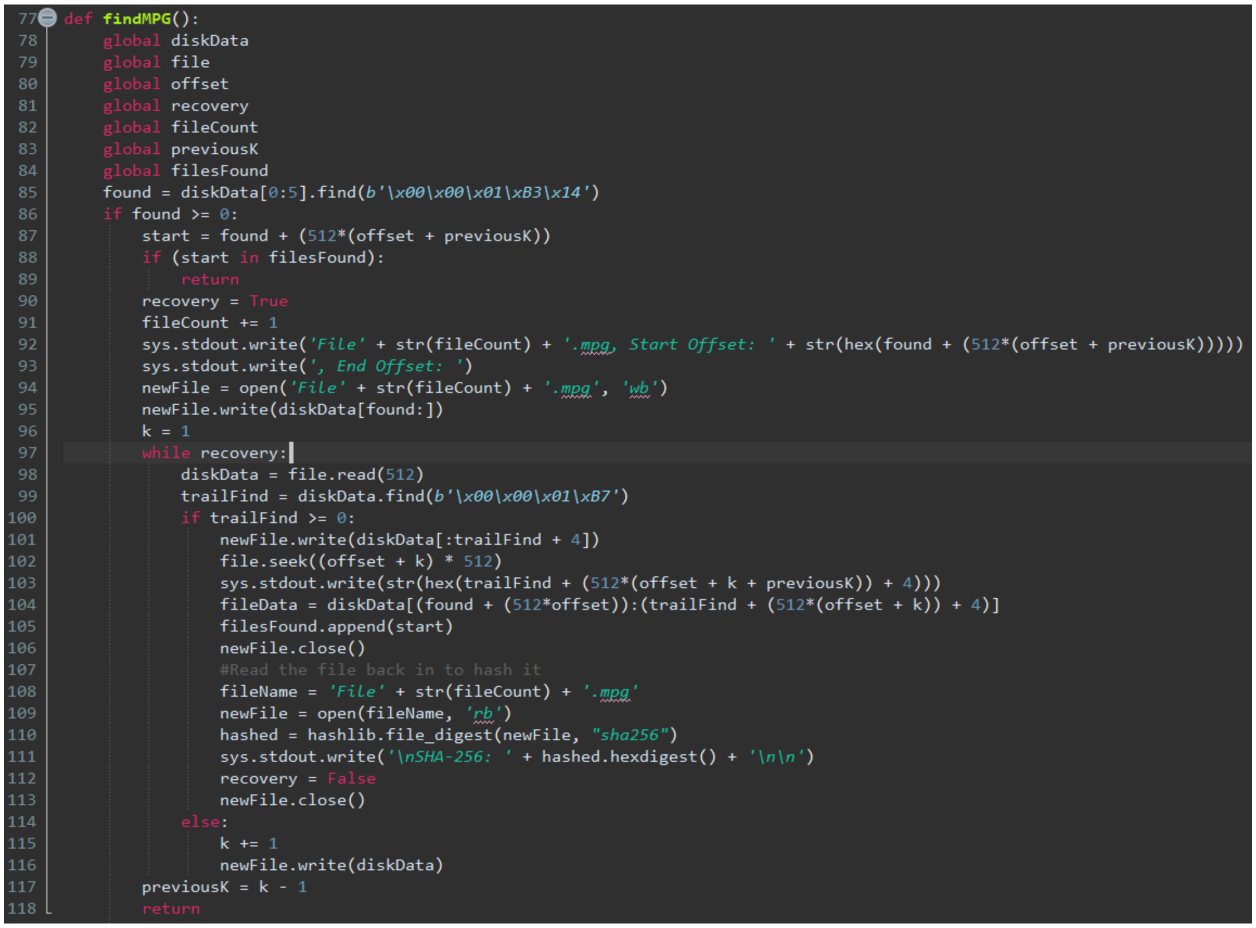
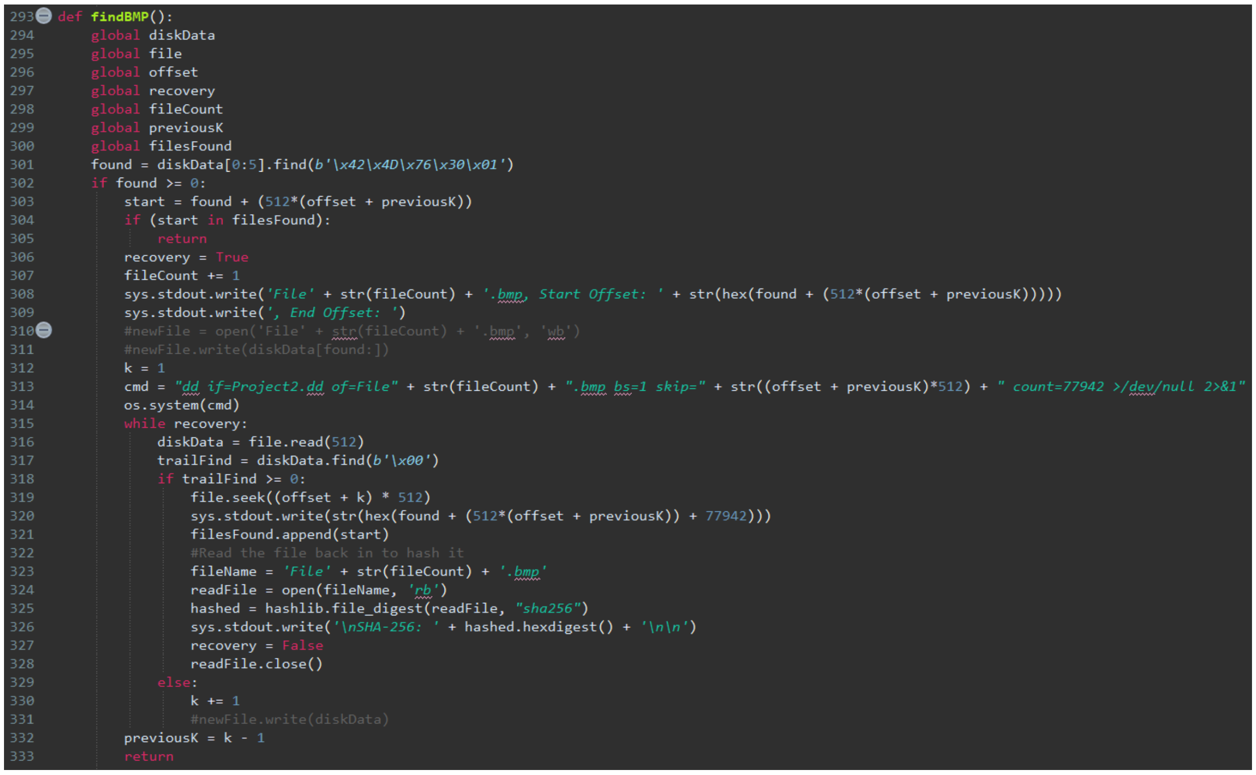

```python
    disk = sys.argv[1]
    file = open(disk, 'rb')
    diskData = file.read(512)

    while diskData:
        findMPG()
        findPDF()
        findBMP()
        findGIF()
        findJPG()
        findDOCX()
        findAVI()
        findPNG()
        #No zip find due to TA saying the zip section
        #has issues and will not be graded
        #findZIP()
        diskData = file.read(512)
        offset += 1
    file.close()
```

Figure 3: Main method of python script that iterates through each 512 byte section of the disk drive searching for files.



```python
def findMPG():
    global diskData
    global file
    global offset
    global recovery
    global fileCount
    global previousK
    global filesFound
    found = diskData[0:5].find(b'\x00\x00\x01\xB3\x14')
    if found >= 0:
        start = found + (512*(offset + previousK))
        if (start in filesFound):
            return
        recovery = True
        fileCount += 1
        sys.stdout.write('File' + str(fileCount) + '.mpg, Start Offset: ' + str(hex(found + (512*(offset + previousK)))))
        sys.stdout.write(', End Offset: ')
        newFile = open('File' + str(fileCount) + '.mpg', 'wb')
        newFile.write(diskData[found:])
        k = 1
        while recovery:
            diskData = file.read(512)
            trailFind = diskData.find(b'\x00\x00\x01\xB7')
            if trailFind >= 0:
                newFile.write(diskData[:trailFind + 4])
                file.seek((offset + k) * 512)
                sys.stdout.write(str(hex(trailFind + (512*(offset + k + previousK)) + 4)))
                fileData = diskData[(found + (512*offset)):(trailFind + (512*(offset + k)) + 4)]
                filesFound.append(start)
                newFile.close()
                #Read the file back in to hash it
                fileName = 'File' + str(fileCount) + '.mpg'
                newFile = open(fileName, 'rb')
                hashed = hashlib.file_digest(newFile, "sha256")
                sys.stdout.write('\nSHA-256: ' + hashed.hexdigest() + '\n\n')
                recovery = False
                newFile.close()
            else:
                k += 1
                newFile.write(diskData)
        previousK = k - 1
        return
```

Figure 4: MPG file recovery, which is the generic recovery process.



```python
def findBMP():
    global diskData
    global file
    global offset
    global recovery
    global fileCount
    global previousK
    global filesFound
    found = diskData[0:5].find(b'\x42\x4D\x76\x30\x01')
    if found >= 0:
        start = found + (512*(offset + previousK))
        if (start in filesFound):
            return
        recovery = True
        fileCount += 1
        sys.stdout.write('File' + str(fileCount) + '.bmp, Start Offset: ' + str(hex(found + (512*(offset + previousK)))))
        sys.stdout.write(', End Offset: ')
        #newFile = open('File' + str(fileCount) + '.bmp', 'wb')
        #newFile.write(diskData[found:])
        k = 1
        cmd = "dd if=Project2.dd of=File" + str(fileCount) + ".bmp bs=1 skip=" + str((offset + previousK)*512) + " count=77942 >/dev/null 2>&1"
        os.system(cmd)
        while recovery:
            diskData = file.read(512)
            trailFind = diskData.find(b'\x00')
            if trailFind >= 0:
                file.seek((offset + k) * 512)
                sys.stdout.write(str(hex(found + (512*(offset + previousK)) + 77942)))
                filesFound.append(start)
                #Read the file back in to hash it
                fileName = 'File' + str(fileCount) + '.bmp'
                readFile = open(fileName, 'rb')
                hashed = hashlib.file_digest(readFile, "sha256")
                sys.stdout.write('\nSHA-256: ' + hashed.hexdigest() + '\n\n')
                recovery = False
                readFile.close()
            else:
                k += 1
                #newFile.write(diskData)
        previousK = k - 1
        return
```

Figure 5: BMP file recovery, which utilizes the "dd" command.

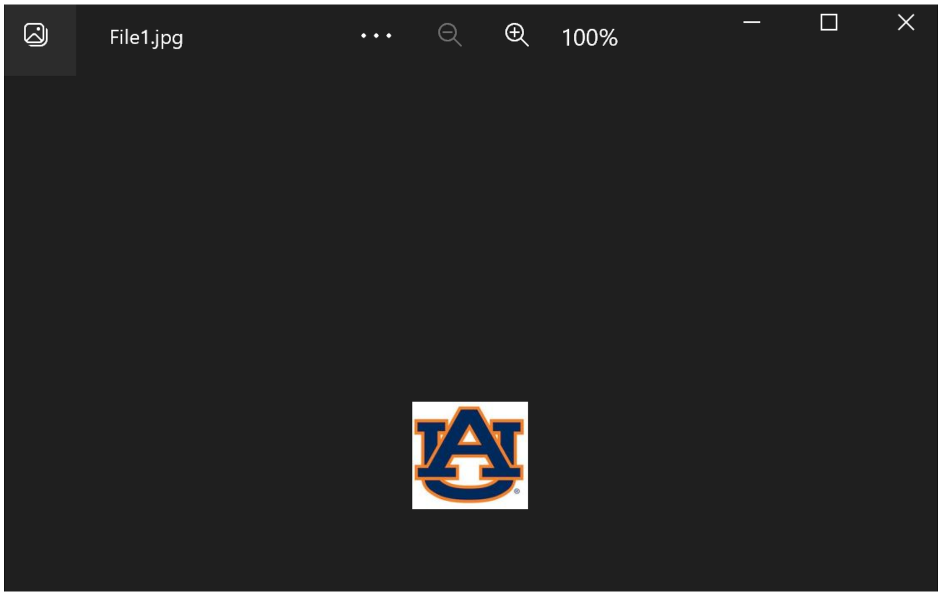


Figure 6: Properly recovered File1.jpg

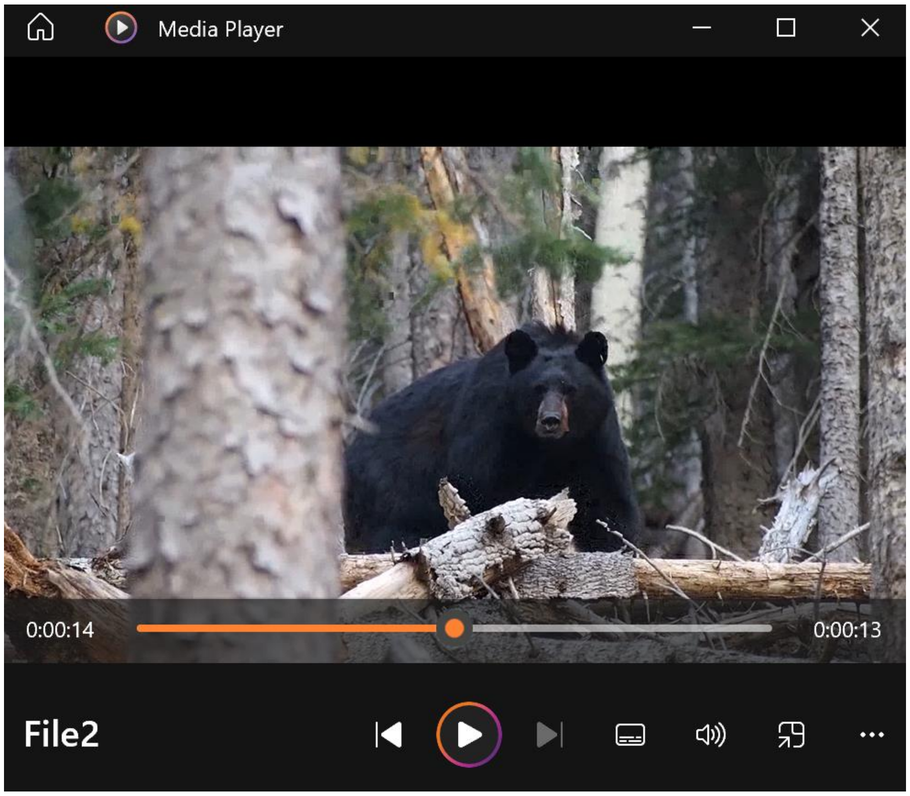


Figure 7: Properly recovered File2.avi

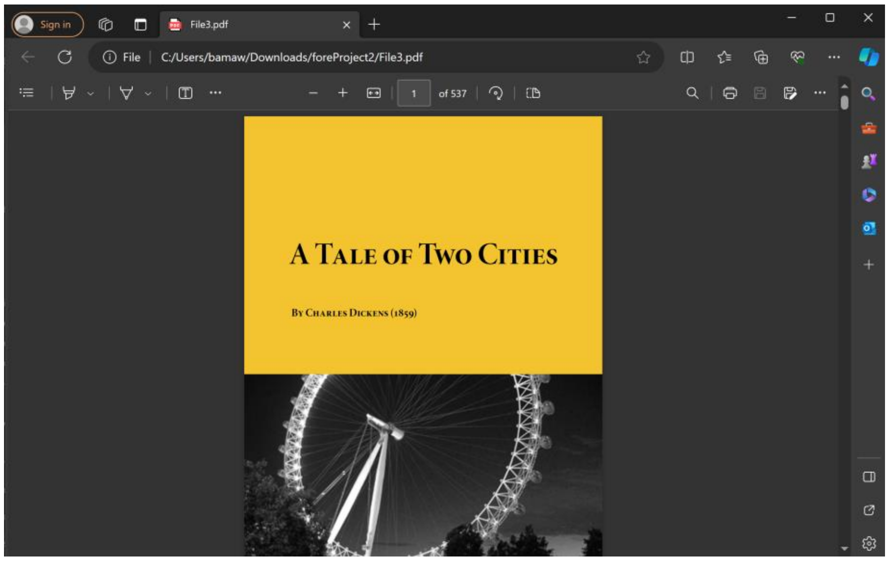

Sign in
File3.pdf
File | C:/Users/bamaw/Downloads/foreProject2/File3.pdf
1 of 537
A TALE OF TWO CITIES
BY CHARLES DICKENS (1859)

Figure 8: Properly recovered File3.pdf

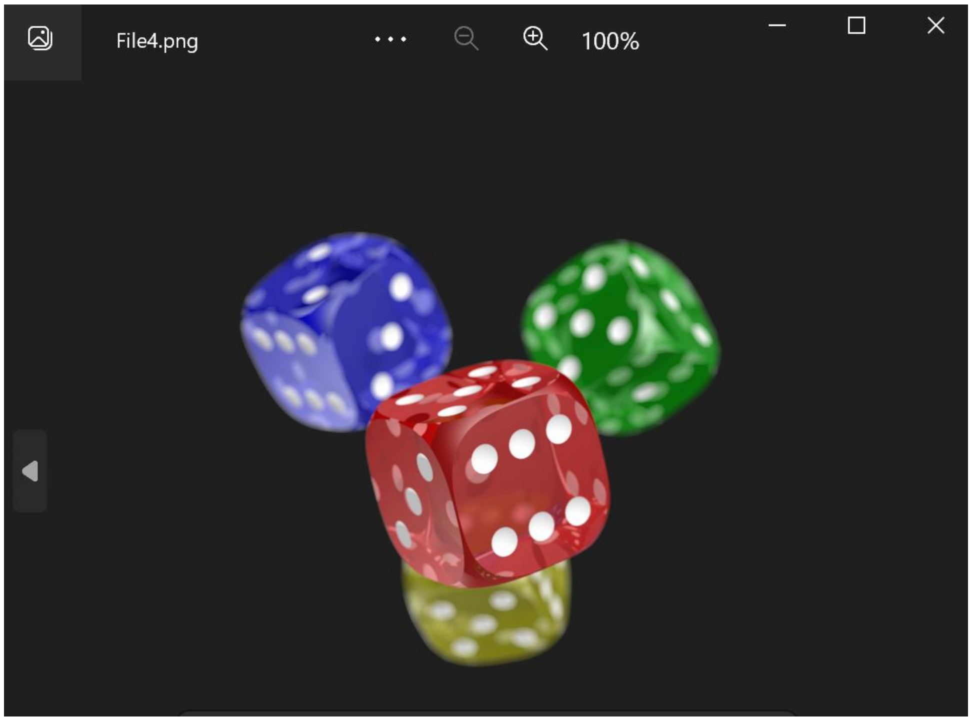


Figure 9: Properly recovered File4.png

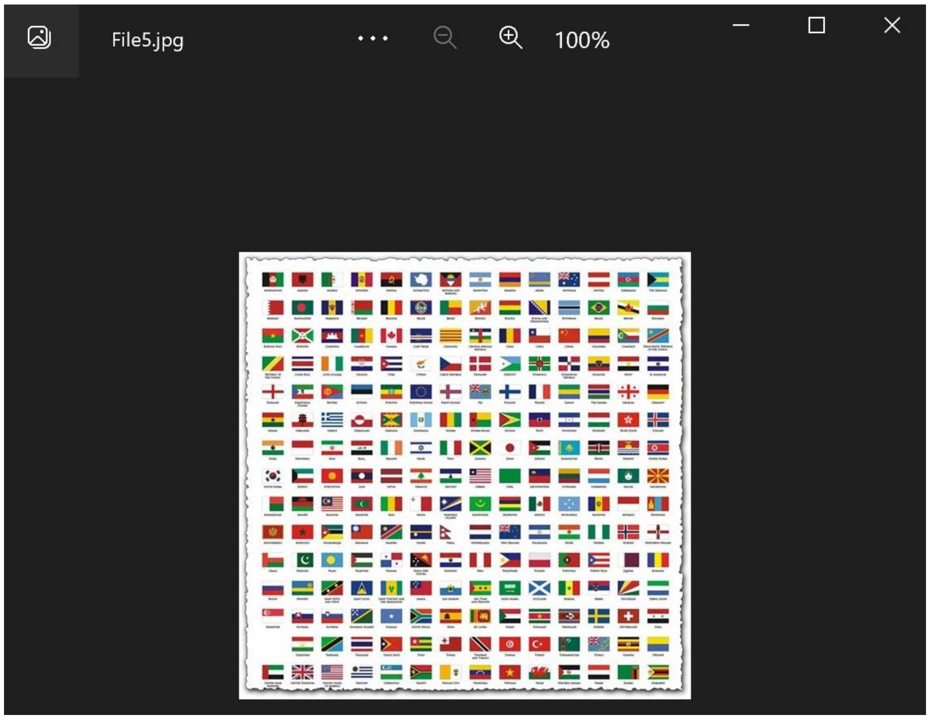


Figure 10: Properly recovered File5.jpg

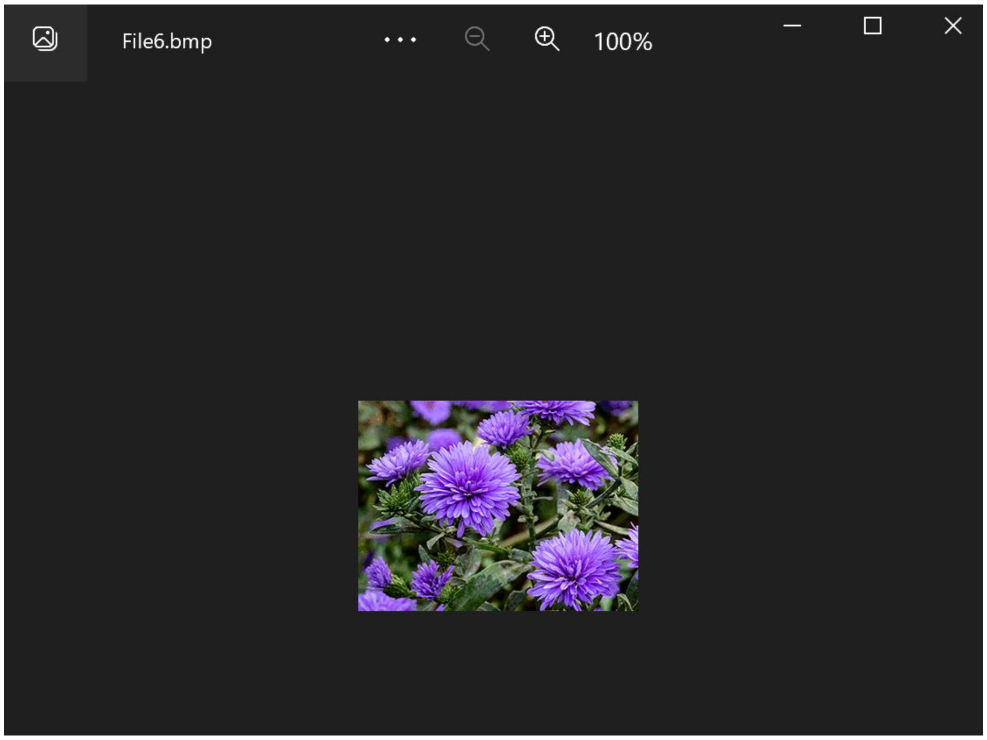
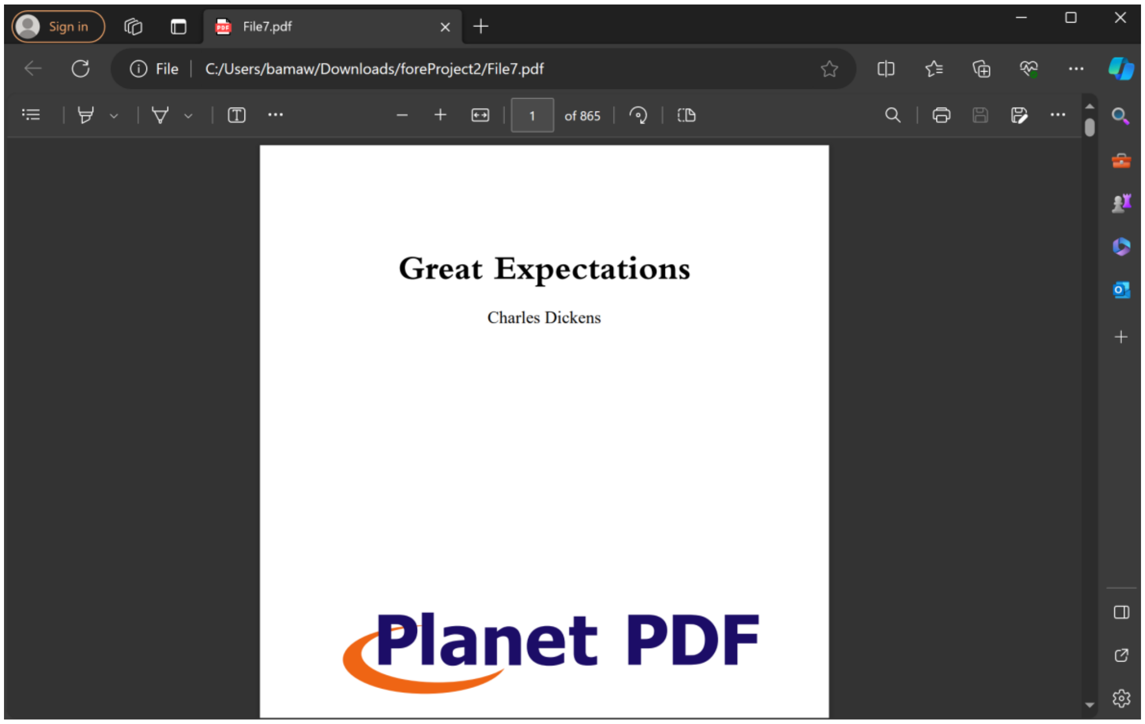


Figure 11: Properly recovered File6.bmp



Great Expectations

Charles Dickens

Planet PDF

Figure 12: Properly recovered File7.pdf

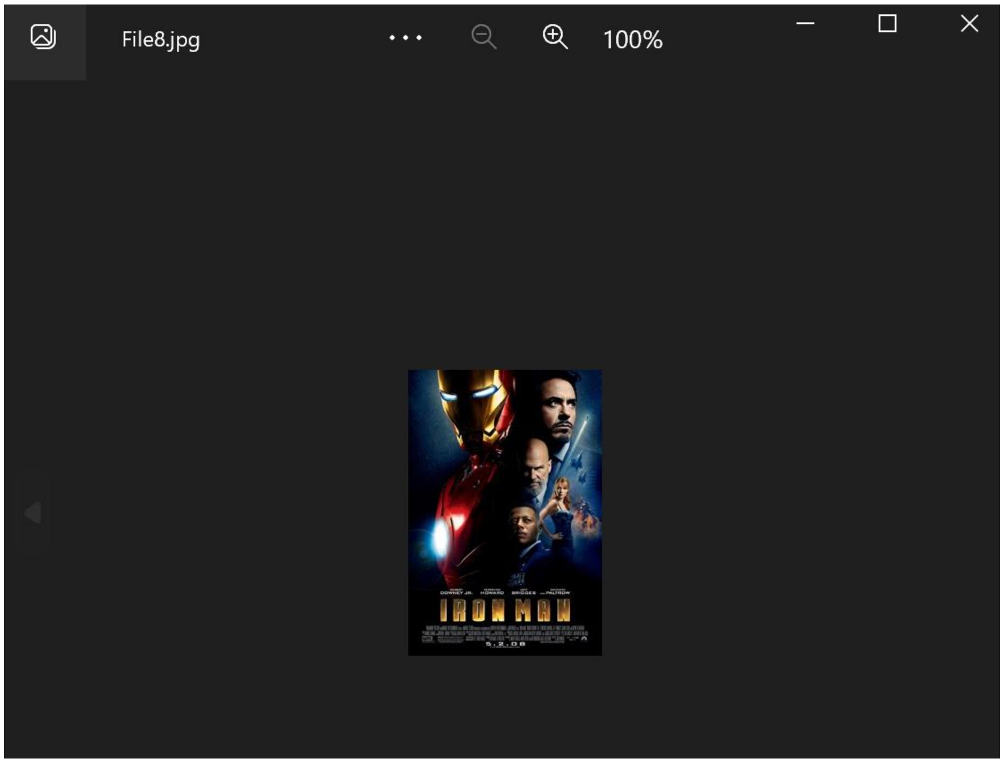


Figure 13: Properly recovered File8.jpg

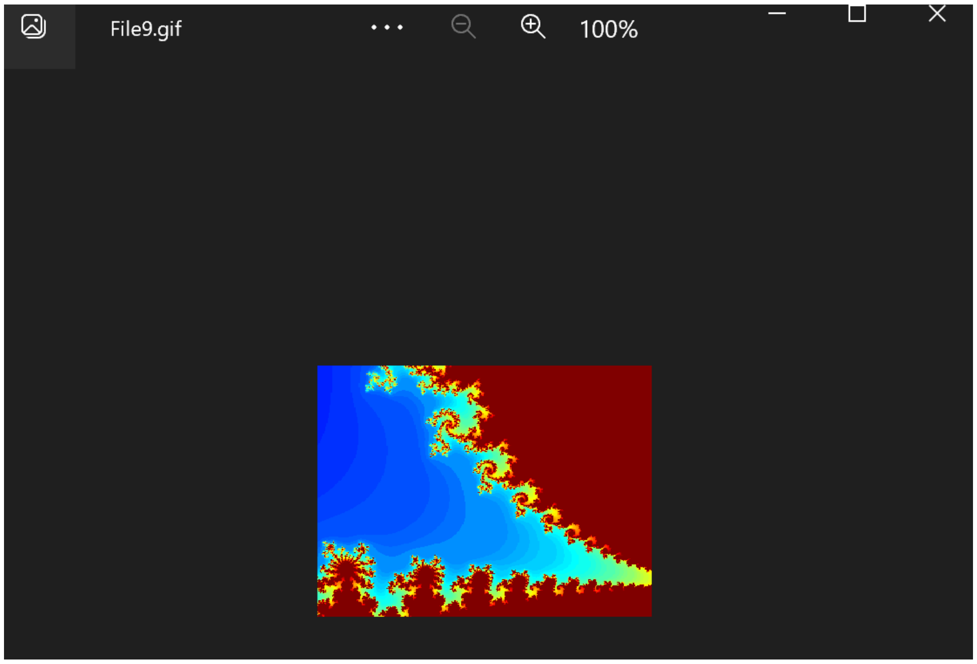


Figure 14: Properly recovered File9.gif

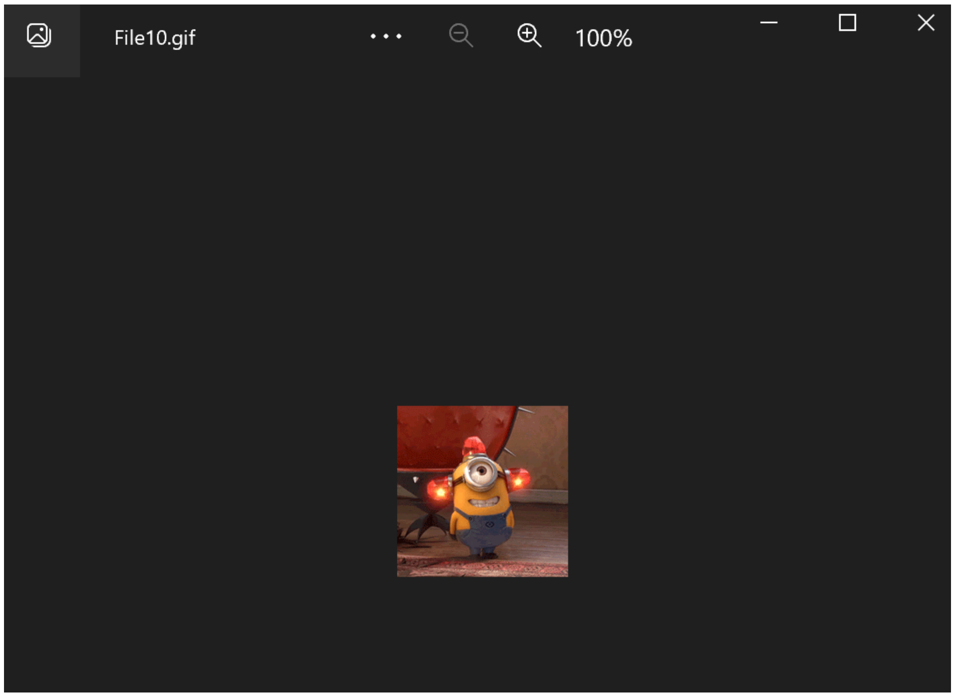


Figure 15: Properly recovered File10.gif

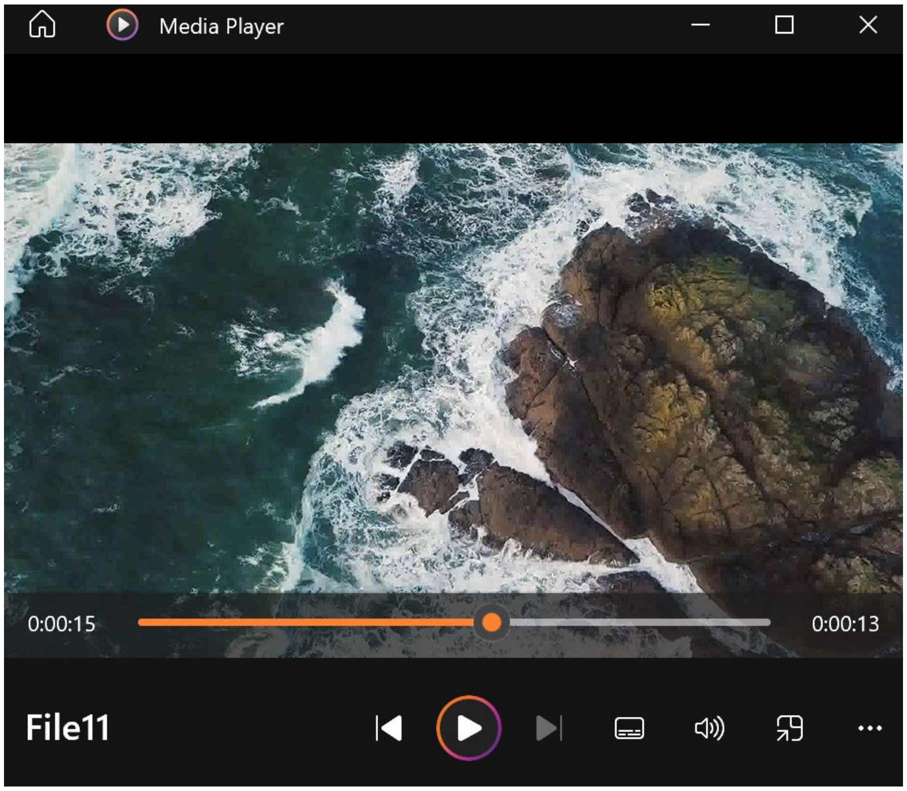
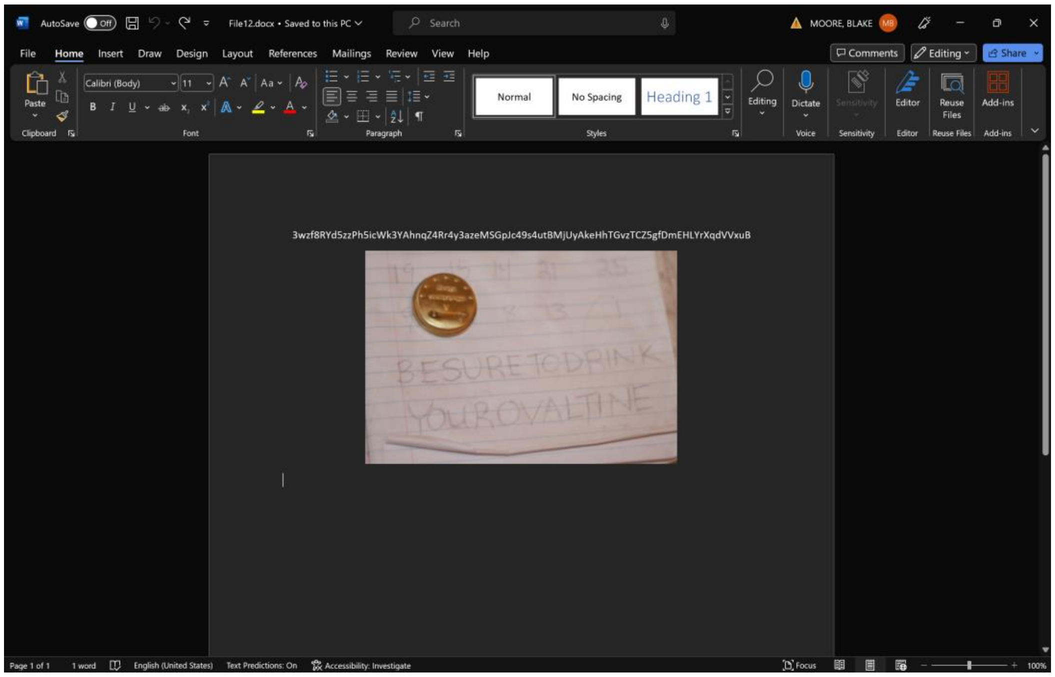


Figure 16: Properly recovered File11.avi



3wzf8RYd5zzPh5icWk3YAhnqZ4Rr4y3azeMSGpJc49s4utBMjUyAkeHhTGvzTCZ5gfDmEHLYrXqdVVxuB

Figure 17: Properly recovered File12.docx

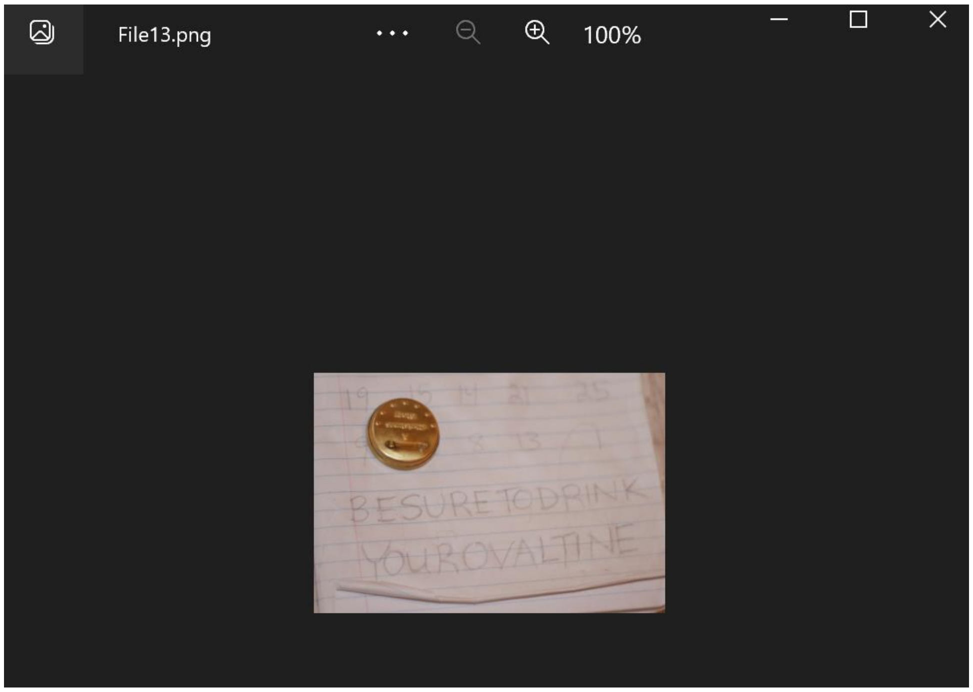


Figure 18: Properly recovered File13.png

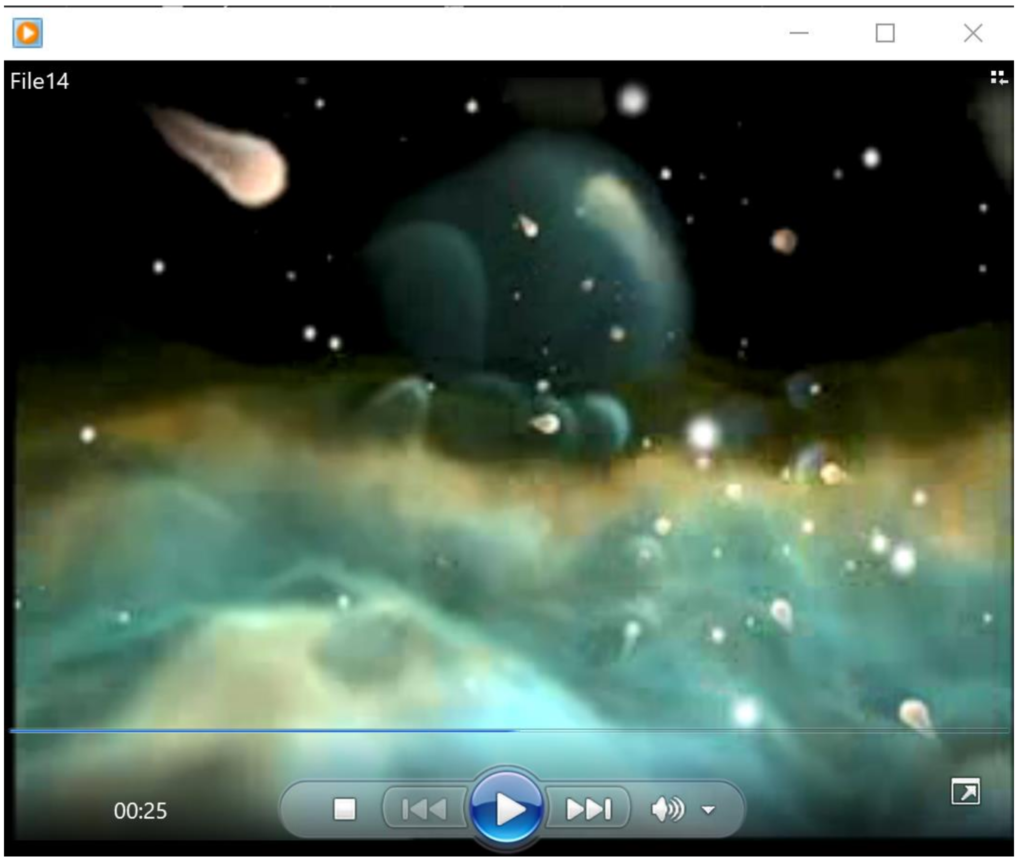
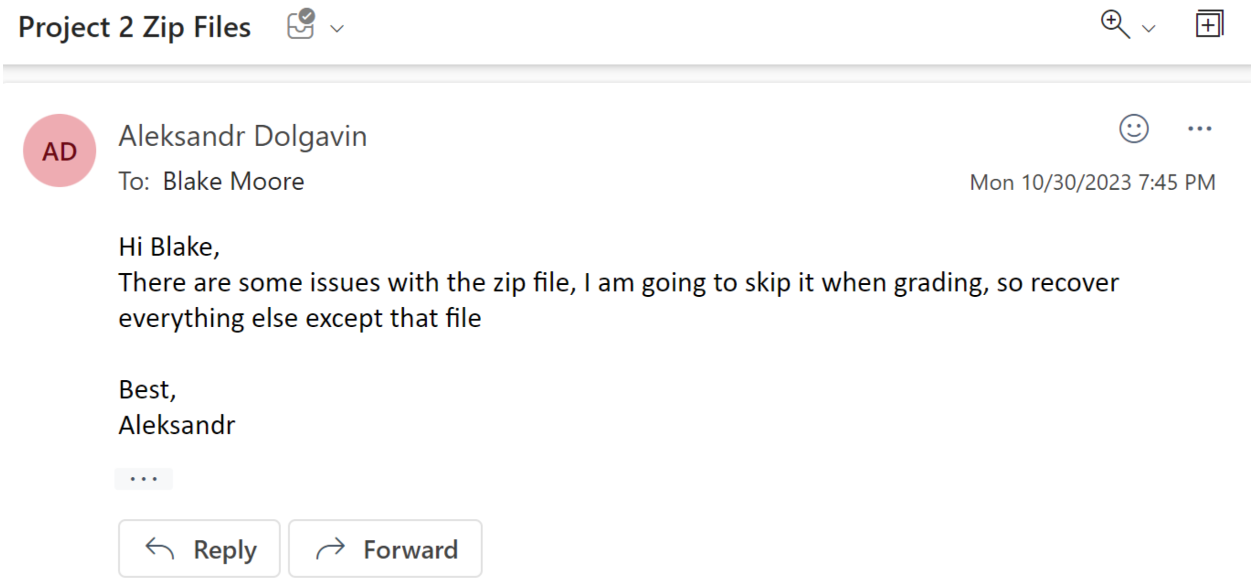


Figure 19: Properly recovered File14.mpg



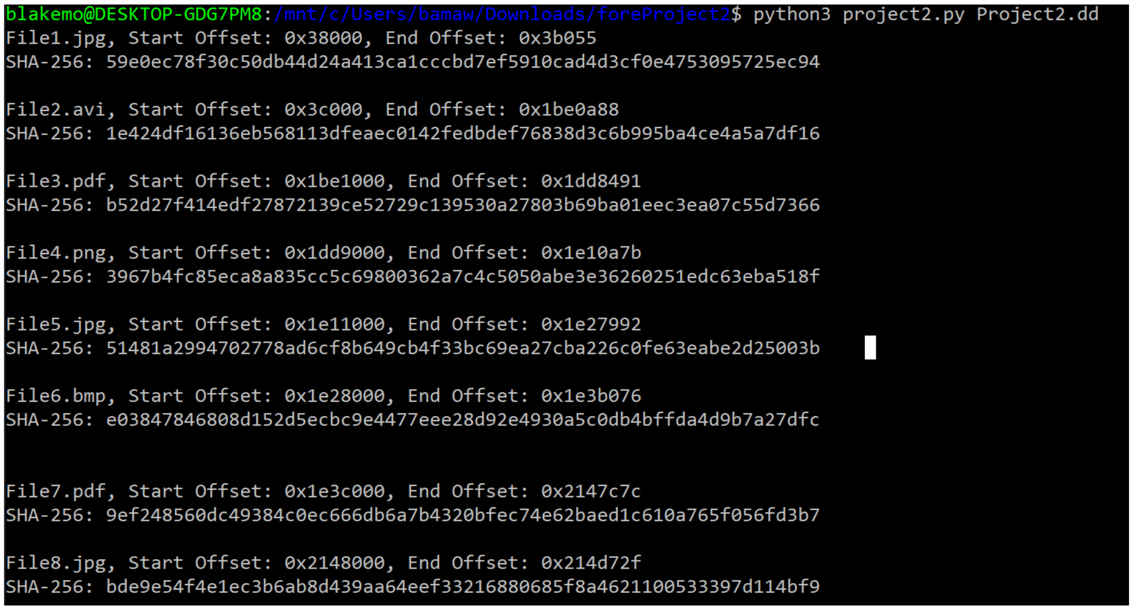
Project 2 Zip Files

Aleksandr Dolgavin
To: Blake Moore
Mon 10/30/2023 7:45 PM

Hi Blake,
There are some issues with the zip file, I am going to skip it when grading, so recover everything else except that file

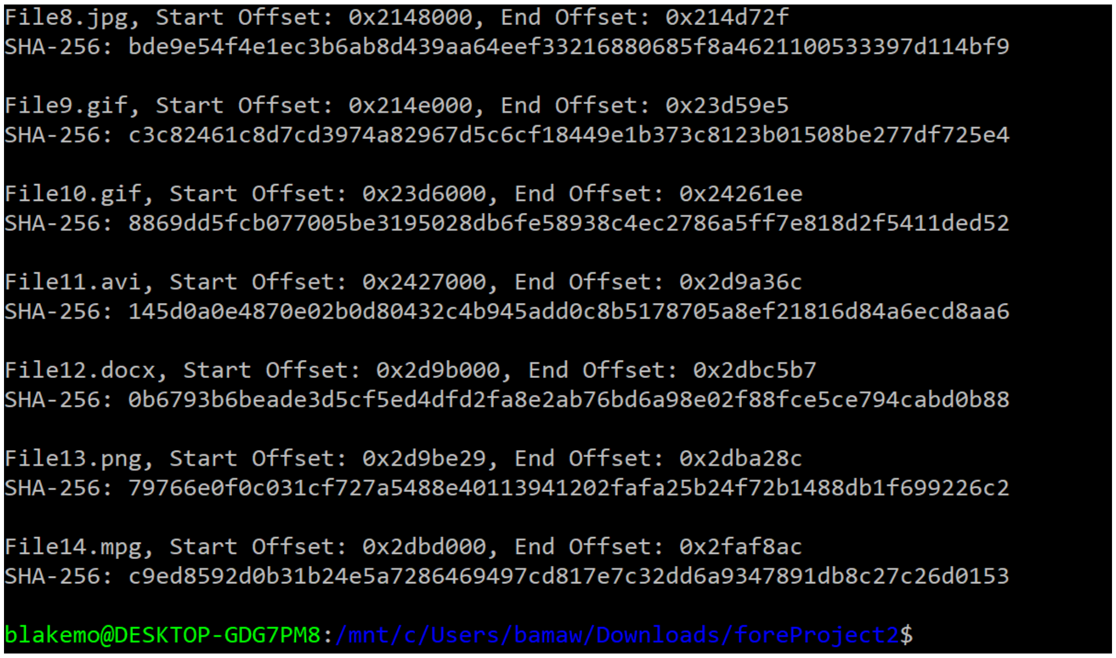
Best,
Aleksandr

Figure 20: TA email detailing issues with .zip files



```
blakemo@DESKTOP-GDG7PM8:/mnt/c/Users/bamaw/Downloads/foreProject2$ python3 project2.py Project2.dd
File1.jpg, Start Offset: 0x38000, End Offset: 0x3b055
SHA-256: 59e0ec78f30c50db44d24a413ca1cccbd7ef5910cad4d3cf0e4753095725ec94

File2.avi, Start Offset: 0x3c000, End Offset: 0x1be0a88
SHA-256: 1e424df16136eb568113dfeaec0142fedbdef76838d3c6b995ba4ce4a5a7df16

File3.pdf, Start Offset: 0x1be1000, End Offset: 0x1dd8491
SHA-256: b52d27f414edf27872139ce52729c139530a27803b69ba01eec3ea07c55d7366

File4.png, Start Offset: 0x1dd9000, End Offset: 0x1e10a7b
SHA-256: 3967b4fc85eca8a835cc5c69800362a7c4c5050abe3e36260251edc63eba518f

File5.jpg, Start Offset: 0x1e11000, End Offset: 0x1e27992
SHA-256: 51481a2994702778ad6cf8b649cb4f33bc69ea27cba226c0fe63eabe2d25003b

File6.bmp, Start Offset: 0x1e28000, End Offset: 0x1e3b076
SHA-256: e03847846808d152d5ecbc9e4477eee28d92e4930a5c0db4bffda4d9b7a27dfc


File7.pdf, Start Offset: 0x1e3c000, End Offset: 0x2147c7c
SHA-256: 9ef248560dc49384c0ec666db6a7b4320bfec74e62baed1c610a765f056fd3b7

File8.jpg, Start Offset: 0x2148000, End Offset: 0x214d72f
SHA-256: bde9e54f4e1ec3b6ab8d439aa64eef33216880685f8a4621100533397d114bf9
```

Figure 21: Output to user, including start/end offsets and hashes



```
File8.jpg, Start Offset: 0x2148000, End Offset: 0x214d72f
SHA-256: bde9e54f4e1ec3b6ab8d439aa64eef33216880685f8a4621100533397d114bf9

File9.gif, Start Offset: 0x214e000, End Offset: 0x23d59e5
SHA-256: c3c82461c8d7cd3974a82967d5c6cf18449e1b373c8123b01508be277df725e4

File10.gif, Start Offset: 0x23d6000, End Offset: 0x24261ee
SHA-256: 8869dd5fcb077005be3195028db6fe58938c4ec2786a5ff7e818d2f5411ded52

File11.avi, Start Offset: 0x2427000, End Offset: 0x2d9a36c
SHA-256: 145d0a0e4870e02b0d80432c4b945add0c8b5178705a8ef21816d84a6ecd8aa6

File12.docx, Start Offset: 0x2d9b000, End Offset: 0x2dbc5b7
SHA-256: 0b6793b6beade3d5cf5ed4dfd2fa8e2ab76bd6a98e02f88fce5ce794cabd0b88

File13.png, Start Offset: 0x2d9be29, End Offset: 0x2dba28c
SHA-256: 79766e0f0c031cf727a5488e40113941202fafa25b24f72b1488db1f699226c2

File14.mpg, Start Offset: 0x2dbd000, End Offset: 0x2faf8ac
SHA-256: c9ed8592d0b31b24e5a7286469497cd817e7c32dd6a9347891db8c27c26d0153

blakemo@DESKTOP-GDG7PM8:/mnt/c/Users/bamaw/Downloads/foreProject2$
```

Figure 22: Continued output to user, including start/end offsets and hashes

6 Conclusions and Recommendations

Using our new python script that leverages file signatures to find and recover files on a disk drive, we were able to recover 14 different files of various common file types. Once recovered, the code hashes the files and displays the results to the user. The process of developing this script required a multifaceted approach wherein we had to employ all the principles of disk image handling and file recovery that we have learned thus far. We hope that our script is robust enough to work with other disk drives and still recover files soundly.