FILE SYSTEM FORENSICS

Assignment One

Dara OSullivan
20080494

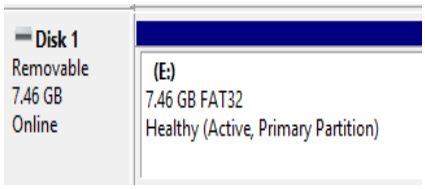
Table of contents

1. Forensics duplicate and evidence verification identifying controls put in place.

For a sound forensics investigation of an evidence file it has to be insured that the file is not tampered with. Duplication of the evidence file and putting controls in place is necessary as to not damage or alter the original file at any rate.

Forensics control: Wiping the USB.

Firstly I checked what disk the usb is on from the disk management. From this I learned it is on disk one. In the command line I moved into the DISKPART drive and selected disk one. From here I deleted the contents of the disk, resulting in it being wiped.




```
C:\>diskpart

Microsoft DiskPart version 10.0.17134.1

Copyright (C) Microsoft Corporation.
On computer: LAPTOP-NHC8DSB5
```

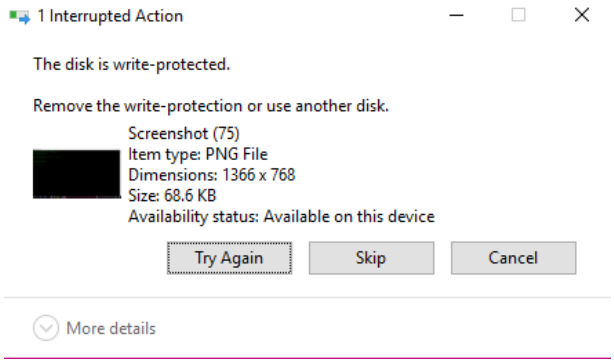
```
DISKPART> select Disk 1

Disk 1 is now the selected disk.
```

Forensics Control: Write Blocking.

The next control put in place was using a Tableau Forensic USB 3.0 Bridge write blocker. A write-block device is a specialized type of computer disk controller made for the purpose of gaining read-only access to drives without the risk of damaging the drive's contents.

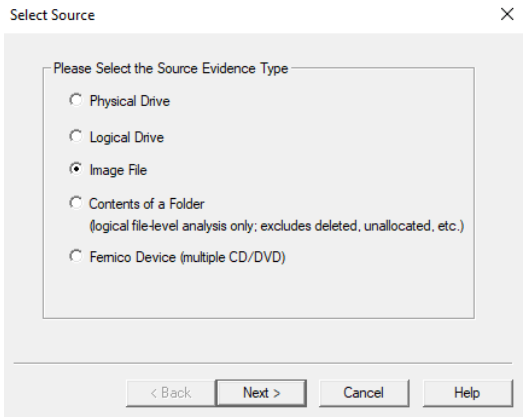
The process began by downloading the file from GitHub onto an empty wiped USB. The Tableau Forensic USB 3.0 Bridge was then setup and the USB was connected. The USB was recognised by the write blocker.

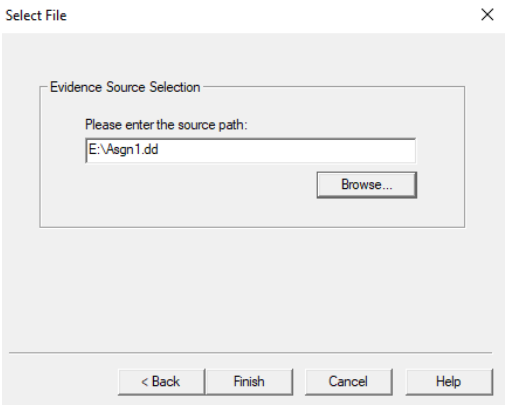
Next, the USB was tested to ensure the process was successful. This was done by trying to move a .png file onto the USB. This action was interrupted as the disk was now write protected ensuring that the process was a success.

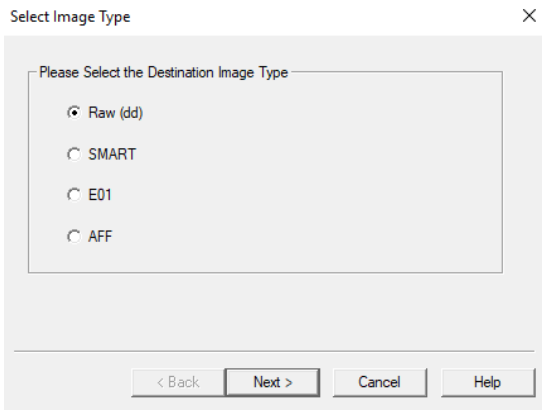


Forensic Duplication of the image file.

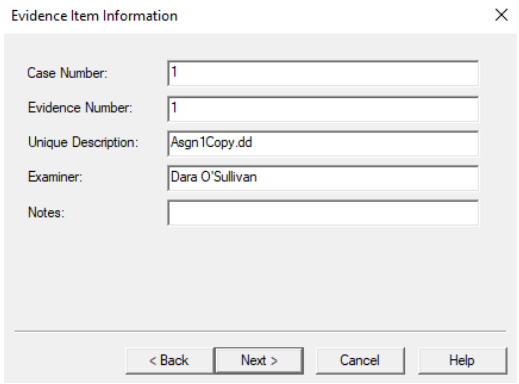
Next a forensic duplicate was made in FTK imager. Firstly, we chose to add an evidence file from the left-hand corner. From there Image file was selected from the different evidence type options. The source path from the image on the USB was given. The image was saved as a raw dd image.

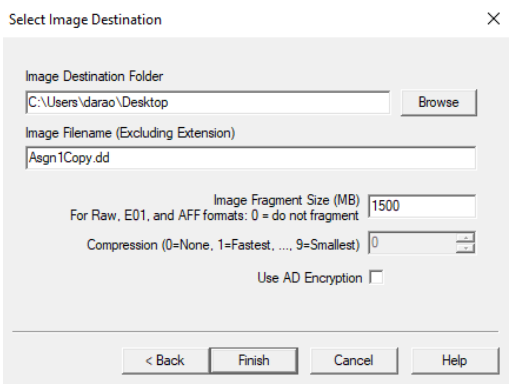






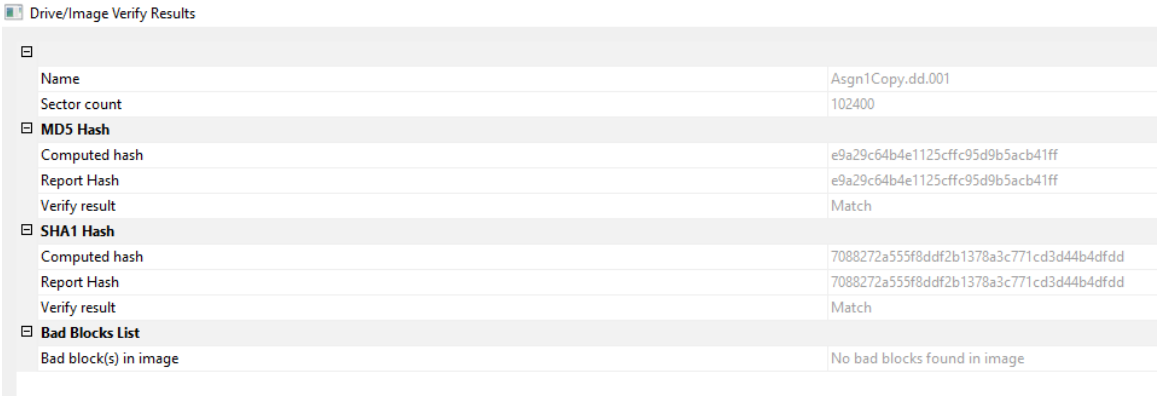
The evidence item information was set and the image duplicate was saved.





Verfication of the image evidence file duplication.

A comparison of the md5 and sha1 hashes of both FTK's results and also those provided in the assignment specification to furhtur ensure sound duplicastion and we see the images are identical.



Drive/Image Verify Results

Name	Asgn1Copy.dd.001
Sector count	102400
MD5 Hash	
Computed hash	e9a29c64b4e1125cffc95d9b5acb41ff
Report Hash	e9a29c64b4e1125cffc95d9b5acb41ff
Verify result	Match
SHA1 Hash	
Computed hash	7088272a555f8ddf2b1378a3c771cd3d44b4dfdd
Report Hash	7088272a555f8ddf2b1378a3c771cd3d44b4dfdd
Verify result	Match
Bad Blocks List	
Bad block(s) in image	No bad blocks found in image

FTK imager image verificaton.

```
Assignment 1 2019 Image Hashes

MD5 Asgn1.dd = e9a29c64b4e1125cffc95d9b5acb41ff

shasum Asgn1.dd = 7088272a555f8ddf2b1378a3c771cd3d44b4dfdd

shasum -a 256 Asgn1.dd =
f509f879b9cc5cb1dcd224b6419b3fd94b4e813119b15a77ee113059d5faaa54
```

..

Assignment specification image hashes.

2. Illustration and naming of where the major data structures required for the examination lie on the analysis image.

The file allocation table (FAT) is a simple file system ideal for smaller disks. It stores entries in a table at the beginning of the volume. A FAT16 file system contains 2 bytes per cluster within the file allocation table. The range of data clusters is between 4087 and 65526 clusters, for FAT16.

The following table shows the order of the data structures that compose a FAT16 disk volume.

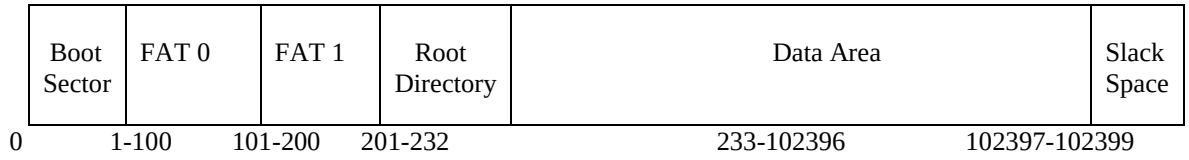
Partition Boot Sector	FAT1	FAT2 (duplicate)	Root folder	Other folders and all files.

Running the Sleuth Kit fsstat command on the evidence file allows for the display of all file system information along with the metadata information.

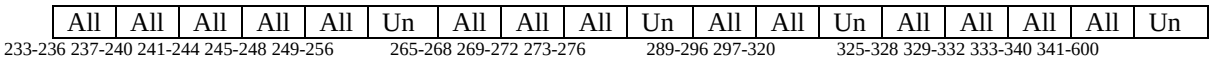
```
File System Type Label: FAT16

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 102399
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 100
* FAT 1: 101 - 200
* Data Area: 201 - 102399
** Root Directory: 201 - 232
** Cluster Area: 233 - 102396
** Non-clustered: 102397 - 102399

METADATA INFORMATION
------------------------------------
Range: 2 - 1635190
Root Directory: 2
```

```
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 25542

FAT CONTENTS (in sectors)
------------------------------------
233-236 (4) -> EOF
237-240 (4) -> EOF
241-244 (4) -> EOF
245-248 (4) -> EOF
249-256 (8) -> EOF
265-268 (4) -> EOF
269-272 (4) -> EOF
273-276 (4) -> EOF
289-296 (8) -> EOF
297-320 (24) -> EOF
325-328 (4) -> EOF
329-332 (4) -> EOF
333-340 (8) -> EOF
341-600 (260) -> EOF
```

A map can be drawn to show clearly the main components of the file. The map below identifies the boot sector, FAT tables, data area, root directory and sectors allocated to files.



Data Area



Above I have indicated the allocated sectors with “All” and unallocated sectors with “Un”.

From the mapping of the data area we see sectors257-264, sectors 277-288, sectors 321-324, and the remainder of the volume are all unallocated.

Then running the Sleuth Kit fls command on our image we see the files in our image.

```
r/r 3:  MYFLOPPY    (Volume Label Entry)
r/r 5:  ._Brief.docx
d/d 7:  .Trashes
d/d 9:  Resources
d/d * 12:       .TemporaryItems
r/r 14: Brief.docx
d/d 16: .Targets
d/d * 17:       _NTITL~1
d/d * 19:       Targets
v/v 1635187:    $MBR
v/v 1635188:    $FAT1
v/v 1635189:    $FAT2
V/V 1635190:    $OrphanFiles
```

From this alone we can tell entry 12, 17 and 19 have been deleted.

To display the node information of these files I used the istat command. From this I gathered the following information.

Entry 5 is located in sectors 289 to 296.

Entry 7 is located in sectors 241 to 244.

Entry 9 is located in sectors 325 to 328.

Entry 12 is located in sector 265 but is of size 0 as it is deleted.

Entry 14 is located in sectors 297 to 320 but the size provided is short 336 bytes from what it needs to be to hold those 24 sectors.

Entry 16 is located in sectors 329 to 332.

Entry 17 and Entry 19 are both given sector 329 and are both of size 0 as they are deleted.

We can now apply this information to out map.

Data Area

All	All	7	All	All	Un	12	All	All	Un	5	14	Un	9	16	All	All	Un
233-236	237-240	241-244	245-248	249-256		265-268	269-272	273-276		289-296	297-320		325-328	329-332	333-340	341-600	

Boot sector	The boot sector is located at the beginning of the volume i.e. the 0th sector. Here information that the file system uses to access the volume is stored. The purpose of analysing the file system category is to determine the file system layout and configuration details. To determine this, we need to process the boot sector data structure. The boot directory tells us a huge amount of extremely useful information for example, the file system type, how many sectors before the file system (which normally tells us the offset), the reserved area, where the FATs are, where the data area is, where the root directory is, sector size, cluster size etc.
File allocation table (FAT)	The file allocation table needs to be stored in a fixed place to allow the files needed to start the system to be allocated. It is an array of 16-bit entries. The file allocation table holds information about each cluster in the volume, essentially a map of the data region. It tells us whether a cluster is unused (0x0000), a bad cluster (0xFFF7), the last cluster in a file (0xFFF8-0xFFFF). The files in the file allocation table are given the first available location in the volume. The starting cluster number is the address of the first cluster used in the file. Each cluster points to the next cluster in the file or it will point to (0xFFFF) to indicate the end of a file. A duplicate of the FAT directly after it in the volume for the sake of redundancy checking. In FAT the data units are

	called Clusters. Each cluster is given an address. The first address is 2.
Root directory	The root directory of the disk contains the number of the first cluster of each file in that directory; the operating system can then traverse the FAT table, looking up the cluster number of each successive part of the disk file as a cluster chain until the end of the file is reached.
Data Area	This is where the actual file and directory data is stored and takes up most of the partition. The size of files and subdirectories can be increased arbitrarily (as long as there are free clusters) by simply adding more links to the file's chain in the file allocation table. FAT uses little-endian format for the entries in the header. The end of the last sector of each FAT copy can be unused if there are no corresponding clusters.

The boot sector, file allocation tables, root directory and the data area are all essential components for the examination on the analysis image. They are all hold necessary information needed to put the pieces together and succeed in an investigation.

3. Decoding of the hexadecimal data for a directory entry. Explanation of the concept of a cluster chain. Following the cluster chain corresponding to the file using its byte offset in the FAT table.

As our file system is FAT16 we work with two bytes at a time which in FAT16 is four numbers and decode using little endian. Using the blkcat -h Asgn1Copy.dd command I displayed the contents of the first data unit in the image in hexdump in the command line and took this to decode. Below the black writing is the original hexdump. The grey writing is after applying little-endian and the red writing is the decimal equivalent.

f0ff ff7f ffff ffff ffff ffff 0700 ffff

fff0 7fff ffff ffff fffff ffff 0007 ffff

x x x x x x x x

0000 0000 ffff ffff ffff 0000 0000 0000

0000 0000 ffff ffff fffff 0000 0000 0000

free free 0 0 0 free free free

1100 ffff 1300 1400 1500 1600 1700 ffff

0011 ffff 0013 0014 0015 0016 0017 ffff

17 0 19 20 21 22 23 0

0000 ffff ffff 1c00 ffff 1e00 1f00 2000

0000 ffff fffff 001c ffff 001e 001f 0020

free 0 0 28 0 30 31 32

2100 2200 2300 2400 2500 2600 2700 2800

0021 0022 0023 0024 0025 0026 0027 0028

33 34 35 36 37 38 39 40

2900 2a00 2b00 2c00 2d00 2e00 2f00 3000

0029 002a 002b 002c 002d 002e 002f 0030

41 42 43 44 45 46 47 48

3100 3200 3300 3400 3500 3600 3700 3800

0031 0032 0033 0034 0035 0036 0037 0038

49 50 51 52 53 54 55 56

3900 3a00 3b00 3c00 3d00 3e00 3f00 4000

57 58 59 60 61 62 63 64

4100 4200 4300 4400 4500 4600 4700 4800

0041 0042 0043 0044 0045 0046 0047 0048

65 66 67 68 69 70 71 72

4900 4a00 4b00 4c00 4d00 4e00 4f00 5000

0049 004a 004b 004c 004d 004e 004f 0050

73 74 75 76 77 78 79 80

5100 5200 5300 5400 5500 5600 5700 5800

0051 0052 0053 0054 0055 0056 0057 0058

81 82 83 84 85 86 87 88

5900 5a00 5b00 5c00 5d00 ffff 0000 0000

89 90 91 92 93 EOF free free

Some useful file signatures in FAT16 are as follows

0000	Free
0002-ffef	Cluster in use
fff0-fff6	Reserved
fff7	Bad cluster
fff8-ffff	Last cluster in particular file

Following the cluster chain corresponding to the file using its byte offset in the FAT table.

We can see from the hex above that 93 is the value before the EOF marker indicating that the last sector used is 93 but we know from our map that there is allocated space up until sector 600. Therefore, the correspondence between the sectors in the file allocation table and the data area is approximately 6.5 times bigger.

Size of the file and the data units it occupies.

The file allocation table is 51200 bytes in size and it occupies data unit 1.

Cluster Chain.

A cluster is simply a fixed length block of bytes allocated for files and directories. The purpose of clusters is to reduce the overhead of managing on-disk data structures. Clusters tend to be groupings of sectors but can be of just one sector. Clusters vary in size depending on the size of the volume and the system involved. Typical cluster size is between 1 sector and 128 sectors.

The starting cluster reference number of a file is listed in the directory area of the device. The file allocation table holds the clusters allocated to one file making the file data trackable. A cluster is the smallest logical amount of disk space that can be allocated to hold a file. As clusters are of a fixed size storing small small files on a filesystem with large clusters will waste disk space, this wasted disk space is called slack space.

In our fat system Fat 16, cluster 2 immediately follows after the last sector of the root directory. The maximum possible size of a FAT16 volume is 2 GB (65526 clusters of at most 64 sectors each).

To calculate the sector address of cluster x

(x-2) * (# of sectors per cluster) + (sector of cluster x)

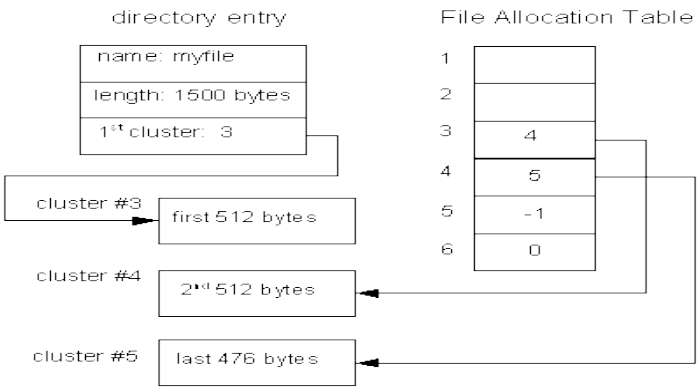
To calculate the cluster address of sector y

((y-sector of cluster 2) (# of sectors per cluster)) + 2

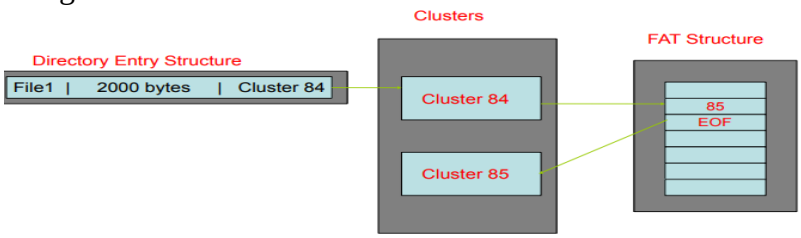
4. Storage and deletion process for a file in a FAT16 file system.

The storage process of a file in a FAT file system

1. Locate the FAT structures, data area and root directory. This can be found in the boot sector.
2. The directory in which you wish to put the file needs to be chosen and located. This can be done by processing through each directory in the root directory searching for the directory name. When the directory is found we will be able to see the starting cluster.
3. Next we process each directory entry in the starting cluster until an unallocated one is found. Setting its name will set its allocation status. The size and current time are also written to the appropriate fields.



4. To allocate clusters for the content we set the next available cluster's entry to EOF. If the file is too large for one cluster then the next available cluster will be allocated also by pushing the EOF value to the following cluster and we set the value of the second cluster being used.



5. In the directory entry we write the address of the starting cluster in the starting cluster field.

The deletion process of a file in a FAT file system.

1. Locate the FAT structures, data area and root directory. This can be found in the boot sector.
2. The directory in which you wish to delete the file from needs to be located. This can be done by processing through each directory in the root directory searching for the directory name just as with storage of a file. When the directory is found we will be able to see the starting cluster.
3. We process the contents of the directory to find a directory entry that has the name of the file we wish to delete. The starting cluster needs to be located.
4. We set the FAT entries for the cluster/clusters to zero.
5. We then set the first byte of the directory entry of file to 0xE5 which will set the directory entry to unallocated.

5. Search and recovery of an allocated file, a deleted image file and a file with a mismatched extension.

An allocated file:

I first ran the Sleuth kit fls command to see the disks files. From this I can see node 14 is an allocated file named Brief.docx. I then did istat on node 14 to get more information. I then used the command icat Asgn1Copy.dd.001 14 > Brief.docx to save this document so I could view it.

```
r/r 3:  MYFLOPPY    (Volume Label Entry)
r/r 5:  ._Brief.docx
d/d 7:  .Trashes
d/d 9:  Resources
d/d * 12:       .TemporaryItems
r/r 14: Brief.docx
d/d 16: .Targets
d/d * 17:       _NTITL~1
d/d * 19:       Targets
v/v 1635187:    $MBR
v/v 1635188:    $FAT1
v/v 1635189:    $FAT2
V/V 1635190:    $OrphanFiles
```

```
Directory Entry: 14
Allocated
File Attributes: File, Archive
Size: 11952
Name: BRIE~1.DOC

Directory Entry Times:
Written:        2019-10-24 15:00:28 (GMT Summer Time)
Accessed:       2019-10-24 00:00:00 (GMT Summer Time)
Created:        2019-10-24 15:00:28 (GMT Summer Time)

Sectors:
297 298 299 300 301 302 303 304
305 306 307 308 309 310 311 312
313 314 315 316 317 318 319 320
```

Viewing the file:

Hey Huey, Louie and Dewey,

I propose that we attack the site in the picture on this disk. We can do this using the resource found in the other picture.

Your Squadron Leader,
Donald

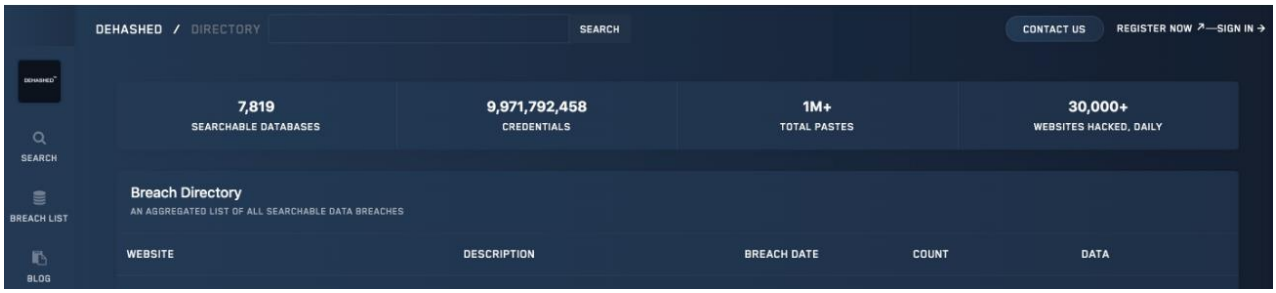
I also viewed this file in Autopsy to see if there was any thing not visible from the document.

```
Hey Huey, Louie and Dewey,

I propose that we attack the site in the picture on this disk. We can do this using the resource found in
 the other picture.

Your Squadron Leader,
Donald


------------------------------METADATA------------------------------

Application-Name: Microsoft Office Word
Application-Version: 16.0000
Author: John Sheppard
Character Count: 157
Character-Count-With-Spaces: 183
Content-Type: application/vnd.openxmlformats-officedocument.wordprocessingml.document
Creation-Date: 2019-10-24T12:22:00Z
Last-Author: John Sheppard
Last-Modified: 2019-10-24T14:00:00Z
Last-Save-Date: 2019-10-24T14:00:00Z
Line-Count: 1
Page-Count: 1
Paragraph-Count: 1
Revision-Number: 1
Template: Normal.dotm
Total-Time: 4
Word-Count: 27
X-Parsed-By: org.apache.tika.parser.DefaultParser
cp:revision: 1
creator: John Sheppard
date: 2019-10-24T14:00:00Z
dc:creator: John Sheppard
dc:publisher:
dcterms:created: 2019-10-24T12:22:00Z
dcterms:modified: 2019-10-24T14:00:00Z
extended-properties:AppVersion: 16.0000
```

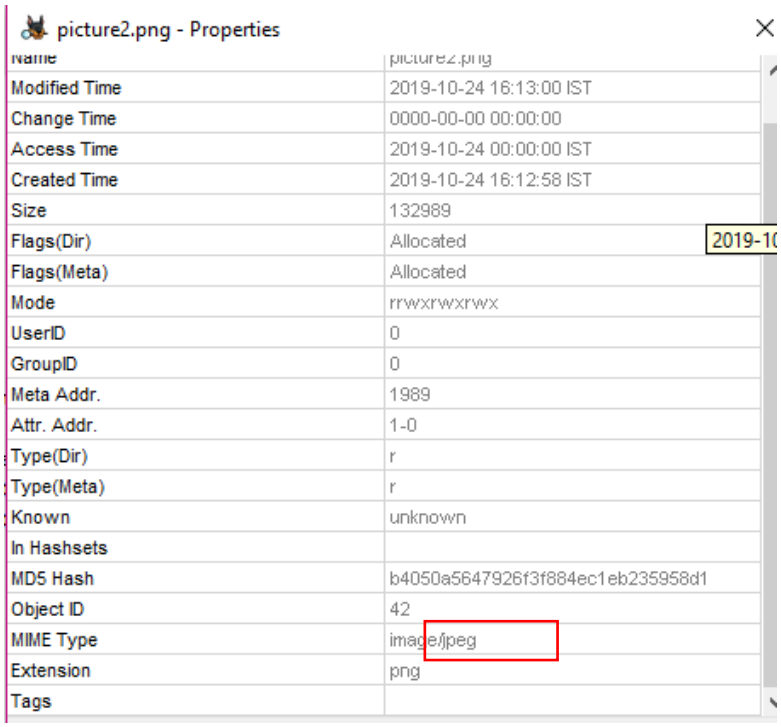
A file with a mismatched extension:

While looking through the files on image in Autopsy I noticed in the Resources folder there was a picture2.png file which we know from fls that it is node 9 of the image and in sectors 325, 326, 327 and 328 from istat.

Later when I used scalpel to find files from the file extensions, scalpel pulled out a .jpg file named 00000000.jpg. When I vied the file I realised it was the same file as the one found in Autopsy.



I then compared the properties of these files and noticed in my picture2.png that the MIME type (media type) of the .png image is actually jpeg.



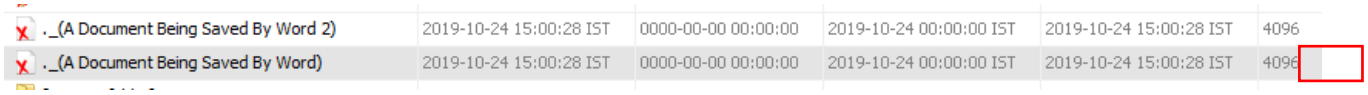
This leads me to believe that the picture2.png is the file with the mix matched file extension.

A deleted image file:

First, I looked at my map which I drew earlier to look at the unallocated space. Then from looking at the size of my unallocated files in autopsy I see a file of eight sectors from sector 257 to 264.

Data Area

All	All	7	All	All	Un	12	All	All	Un	5	14	Un	9	16	All	All	Un
233-236	237-240	241-244	245-248	249-256		265-268	269-272	273-276		289-296	297-320		325-328	329-332	333-340	341-600	



From the command line I then used the dd command to save the contents of these sectors to a .doc file. The file contained these two snippets, which may be of use.

```
Mac OS X
ATTR;
com.apple.quarantine0086;00000000;Microsoft Word;
This resource fork intentionally left blank
```

We can also see this document in FTK imager and Autopsy.

Other files I found while looking at this image:

. _~WRD0000:

```
Mac OS X
WXBNMSWD
ATTR;
com.apple.quarantine
com.apple.lastuseddate#PS
$com.apple.metadata:_kMDItemUserTags
0082;5dblae7c;Microsoft Word;|
bplist00
```

. _~WRL0001

```
Mac OS X
ATTR;
com.apple.quarantine
com.apple.lastuseddate#PS
$com.apple.metadata:_kMDItemUserTags
7com.apple.metadata:kMDLabel_doc54n43osyriip2yklmyadinm
0082;5db1bf7b;Preview;|
bplist00
V]!con
E:We`
;3j~
```

Fseventsd-uuid:

```
C1BF1D81-12A3-48BB-83AB-69AC7C702A95


------------------------------METADATA------------------------------

Content-Encoding: ISO-8859-1
Content-Type: text/plain; charset=ISO-8859-1
X-Parsed-By: org.apache.tika.parser.DefaultParser
```

Questions:

1. Who is involved in the hack?
   Huey, Louie, Dewey and Donald are involved in the attack.
2. What site has been targeted?
   Apple.com
3. How could they attack the targeted site?

Possibly with the use of user tags.

4. What have they done to mask the files on the disk?

   They have deleted files and used mix matched extensions on files.