# COMPUTER FORENSICS

## INVESTIGATION INTO FAT12 FILE SYSTEM

EOIN DALTON
20070289

# Contents

# 1. Tools Used for Analysis

To perform the analysis on the FAT32 file system image there will be two tools used, the first a command line tool called SleuthKit which is a forensic investigating tool and is a free install found at: https://www.sleuthkit.org/. The second is a graphical interface tool call FTK imager found at https://marketing.accessdata.com/ftk-imager-3.4.3-download . SleuthKit will be used through an Ubuntu virtual machine and FTK imager will be used on Windows 10 host machine. Most of the work will be done using SleuthKit through the command line interface. FTK imager will be user to verify the analysis that has been just done.

# 2. Securing the Evidence

Date Received: 27/02/2019 Time: 13:15

Type: FAT32 File System Format: image

Received By: Eoin Dalton Identification: 20070289

In order to keep the evidence from being corrupted by the investigating machine it is a good idea to store the image on a secure location like a clean folder or USB drive. The evidence that was obtained in this case will be stored on a USB. But before we store the image on a USB, we must make sure that the USB is completely clean. To see of the USB is clean the SleuthKit was used. If you refer to figure 1 you will see that the USB contained to files.

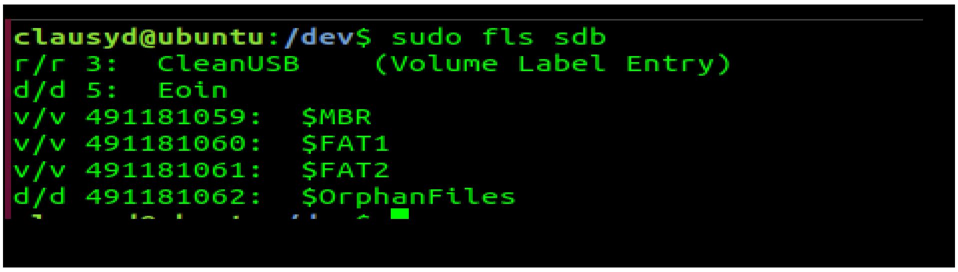



*Figure 1: USB used to store evidence.*

Because the USB contained files it needed to be fully wiped. For it to be completely wiped it must be rewritten with ones and zeros. This was done with the following command:

**sudo dd if=/dev/sdb/zero of=/dev/sdb bs1k count=2048**

If you refer to figure 2 you will see that the wipe was successful. After this a new file system must be added to the USB stick and this command completed wipe everything.

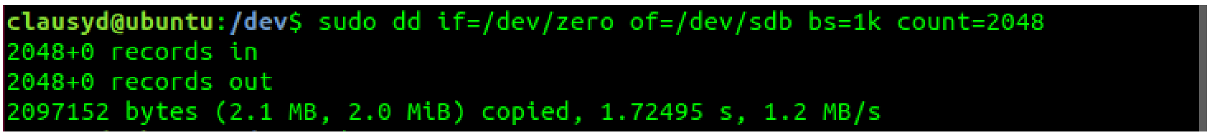



*Figure 2: Rewriting 1's and 0's to USB.*

In order to add a new file system a Linux tool called gparted was used. This is a GUI tool that easily allows users to write file system to devices. If you refer figure 3 you will see the USB sdb with the new FAT32 file system after being written to the device.

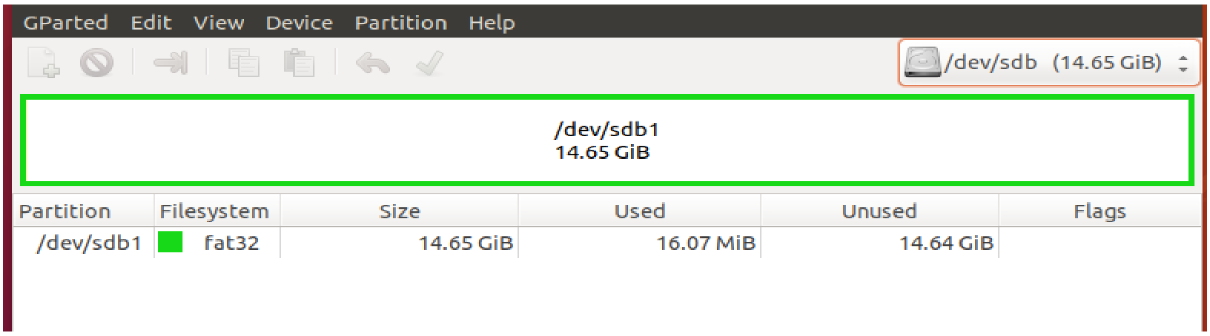



*Figure 3: Writing new FAT32 file system with gparted.*

If you refer to figure 4 you will see a listing of the USB stick and the only file contained on it is the evidence.

```
clausyd@ubuntu:/dev$ sudo fls sdb1
r/r 5:  FSF-Asgn1-18.dd
v/v 491138051:  $MBR
v/v 491138052:  $FAT1
v/v 491138053:  $FAT2
d/d 491138054:  $OrphanFiles
```

*Figure 4: Clean USB stick with evidence on it.*

## 2.1 Verifying image

After receiving the evidence, the first step was to verify the image hash value that we received via GitHub as seen in figure 5. This is done to verify that this is actually the evidence and that someone hasn't switched the evidence with something that could be clean.

```
Assignment 1 2018 Image Hashes

MD5 (FSF-Asgn1-18.dd) = 8ae98281ba211f24294f0ee28d1886d8
shasum FSF-Asgn1-18.dd = 2dfcd2c9fe0f8c16615b9738ac6cfc52e32b8540
shasum -a 256 FSF-Asgn1-18.dd =
bc8487c08e930c2ade25ac5098f85cb003b6ecb805aaf831bd4cd028a4c81643
```

*Figure 5: Original Hash Values Located on GitHub.*

The following commands where used to generate the has values.

**md5sum FSF-Asgn1-18.dd**

8ae98281ba211f24294f0ee28d1886d8  FSF-Asgn1-18.dd

**shasum FSF-Asgn1-18.dd**

2dfcd2c9fe0f8c16615b9738ac6cfc52e32b8540  FSF-Asgn1-18.dd

**sha256sum FSF-Asgn1-18.dd**

bc8487c08e930c2ade25ac5098f85cb003b6ecb805aaf831bd4cd028a4c81643  FSF-Asgn1-18.dd

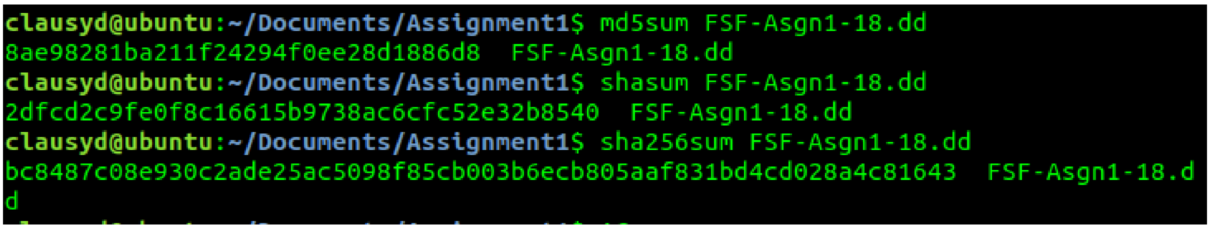



*Figure 6: Hash values generated on investigating machine.*

## 2.2 Making Forensic Duplicates

Once the image was verified it was then time to make a forensic copy for investigator to perform their analysis. This is done so that the original evidence isn't compromised in anyway. To make the copy of the image the following command was used:

**dd if=FSF-Asgn1-18.dd of=copy-FSF-Asgn1-18.dd**

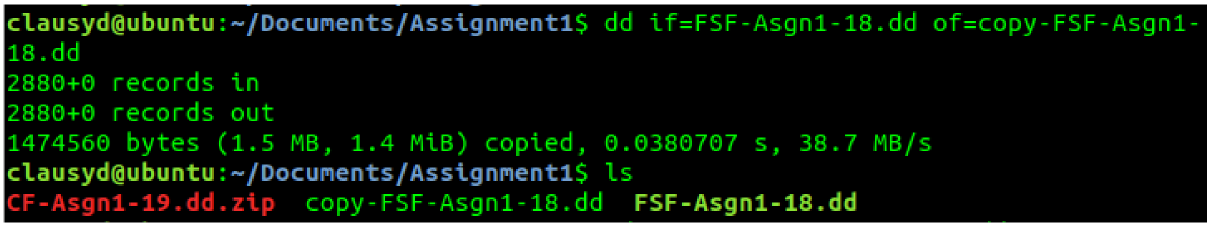



*Figure 7: Making the forensic image on the investigating machine.*

Once the copy was made the investigator then varied that hash values of the copied version matched the hash value of the original image. The hash values are taken on the content contained within the file and shouldn't change no matter how many copies are taken. This is a way of making sure that the evidence wasn't altered in any shape or form by the person performing the analysis. The following command where used to generate the has values:

**md5sum copy-FSF-Asgn1-18.dd**

8ae98281ba211f24294f0ee28d1886d8  copy-FSF-Asgn1-18.dd

**shasum copy-FSF-Asgn1-18.dd**

2dfcd2c9fe0f8c16615b9738ac6cfc52e32b8540  copy-FSF-Asgn1-18.dd

**sha256sum copy-FSF-Asgn1-18.dd**

bc8487c08e930c2ade25ac5098f85cb003b6ecb805aaf831bd4cd028a4c81643  copy-FSF-Asgn1-18.dd

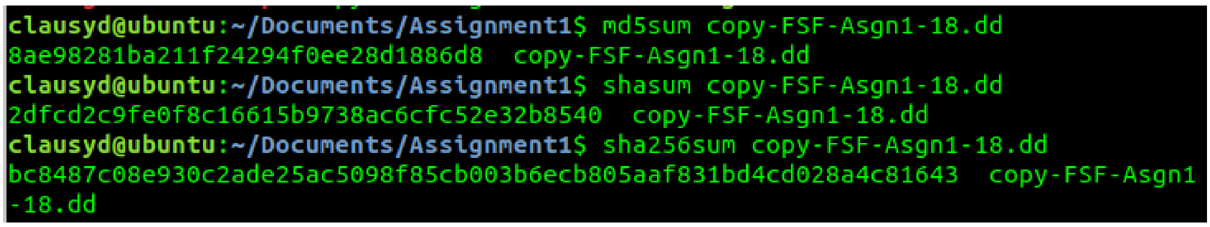



*Figure 8: Generating the hash values of the copied version of the image.*

```
clausyd@ubuntu:~/Documents/Assignment1$ md5sum FSF-Asgn1-18.dd
8ae98281ba211f24294f0ee28d1886d8  FSF-Asgn1-18.dd
clausyd@ubuntu:~/Documents/Assignment1$ shasum FSF-Asgn1-18.dd
2dfcd2c9fe0f8c16615b9738ac6cfc52e32b8540  FSF-Asgn1-18.dd
clausyd@ubuntu:~/Documents/Assignment1$ sha256sum FSF-Asgn1-18.dd
bc8487c08e930c2ade25ac5098f85cb003b6ecb805aaf831bd4cd028a4c81643  FSF-Asgn1-18.d
d
```

*Figure 9: Original hash values of image.*

If you look at figures 8 and 9 you will see that all the hash values are matching, and the evidence hasn't been compromised in any way.

## 3. FAT12 File System Data Structures

The next step taken while investigating the evidence was the layout the structure of the file system. To attain the information the following command that was issued was:

**fsstat copy-FSF-Asgn1-18.dd**

and the output from this command is seen in figure 10.

```
clausyd@ubuntu:~/Documents/Assignment1$ fsstat copy-FSF-Asgn1-18.dd
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT12

OEM Name: WINIMAGE
Volume ID: 0x50e425fc
Volume Label (Boot Sector):
Volume Label (Root Directory):
File System Type Label: FAT12

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 - 2879
* Reserved: 0 - 0
** Boot Sector: 0
* FAT 0: 1 - 9
* FAT 1: 10 - 18
* Data Area: 19 - 2879
** Root Directory: 19 - 32
** Cluster Area: 33 - 2879
```

```
METADATA INFORMATION
--------------------------------------------
Range: 2 - 45782
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 512
Total Cluster Range: 2 - 2848

FAT CONTENTS (in sectors)
--------------------------------------------
35-199 (165) -> EOF
200-200 (1) -> EOF
201-452 (252) -> EOF
```

*Figure 10: FAT32 file system data structure.*

Once this was done the investigator was then able to layout the data structure as referred to in figure 11 in a more readable fashion with all the partitions and their sizes clearly identified. This was used as a point of reference throughout the investigation.

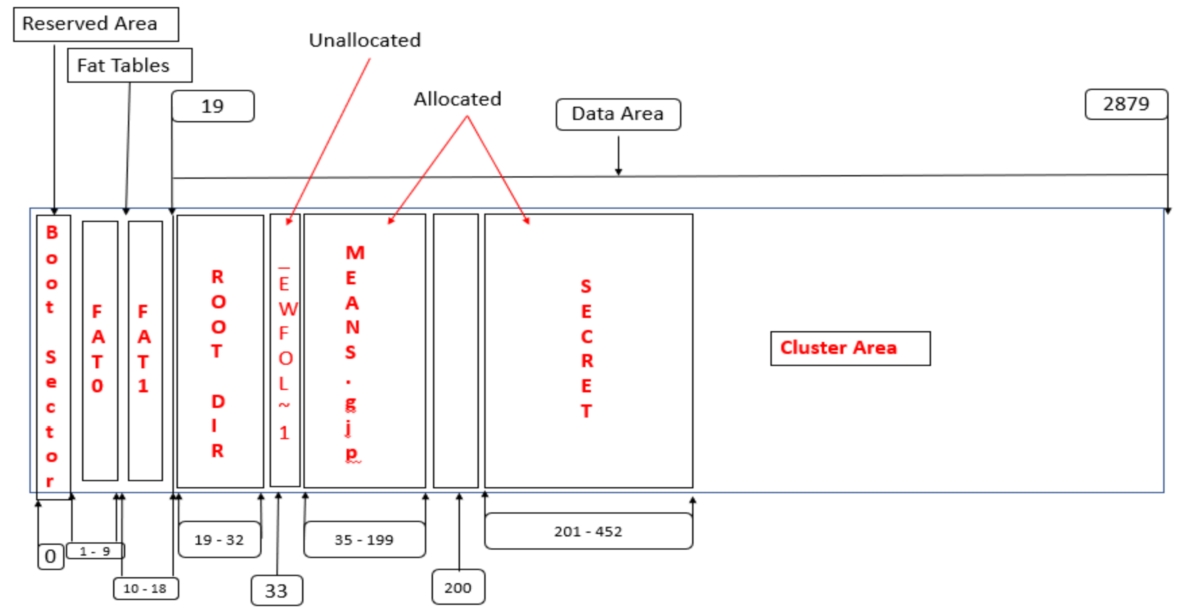



*Figure 11: Data structure mapped out.*

# 4. Decoding Hexadecimal Data

## 4.1 Cluster Chain

Referred to in figure 12 is the process of a cluster chain. The cluster chain will start in the root directory. When a user creates a new directory the boot sectors will search for the different structure contained within the Fat file system such as the fat structures, data area and the root directory.

Once the root directory is located it will search for an empty slot. Once it finds one it will place the name of the directory in the empty slot.

Next a user creates a new text file and places the file in the directory. Then the file is processed and searches for an available cluster. In figure 12 cluster 90 was available so that entry is places in the root directory. The name of the file and some other detail such as time created, and the file size is placed in the cluster number 90.

Next the content of the file is processed and again it searches for an available cluster to put the content in. In figure 12 the next available slot is 200 so that number is placed in the cluster with the name, time and file size.

The last step is where the Fat table are updated with the information. The boot sector will search the Fat table for an available slot and once it finds one it places the number of the cluster where the contents of the file are stored. This is different in figure 12 as the contents of the file is bigger than the size of the cluster. In this case the remaining contents of the file a placed in the next cluster which is 201. This value is then placed in cluster 200 in the Fat tables. This is done because when the Fat table are read by the boot sector it goes to 200 and that tell the boot sector that there are more contents of the file found at cluster 201. In cluster 201 of the Fat tables the value EOF is place which signals end of the file.

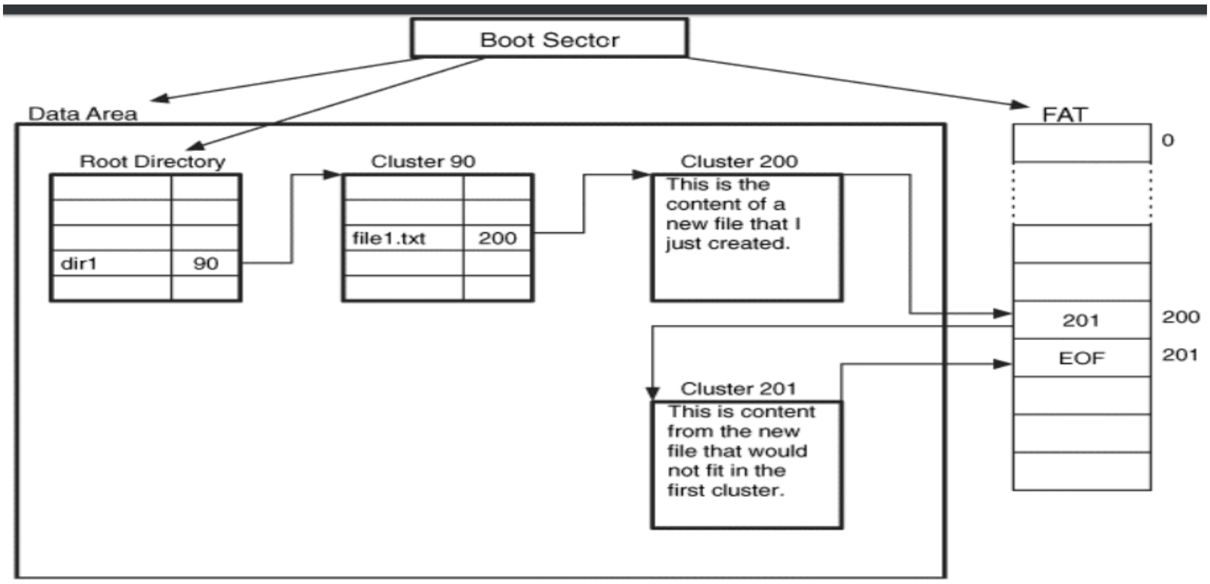



*Figure 12: Cluster chain for file creation (Sheppard, 2019).*

Let's look an identifying the cluster chain on the evidence this investigation is looking at. Referred to in figure 12. This is the root directory found in sector 19. We no this because we laid out our data structure earlier shown in figure 11 where the root directory runs from 19-32. If you refer back to figure 11 this is the layout of a Fat12 file is a file system. To identify all the parts of the file a tool called Active Disk Editor was used. This is an open source tool for Windows and Linux OS's. Please pay attention to the first cluster (high word) it is 0004 this converted to a decimal value returns 4. The reason for this is the root directory span form 19-32 as stated above. This is an offset of 32 so the first cluster actually starts at 33. If you refer back to figure 11 you will see that the MEANS.zip is start at 35. The folder starts at 35 and the file inside the folder starts at 36.

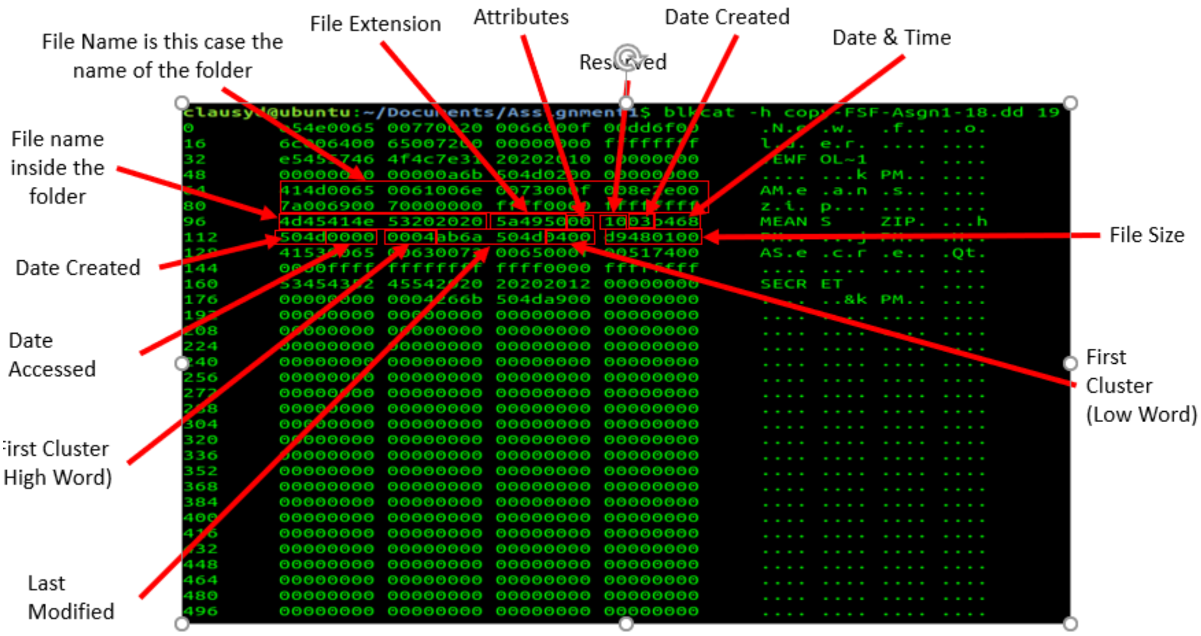
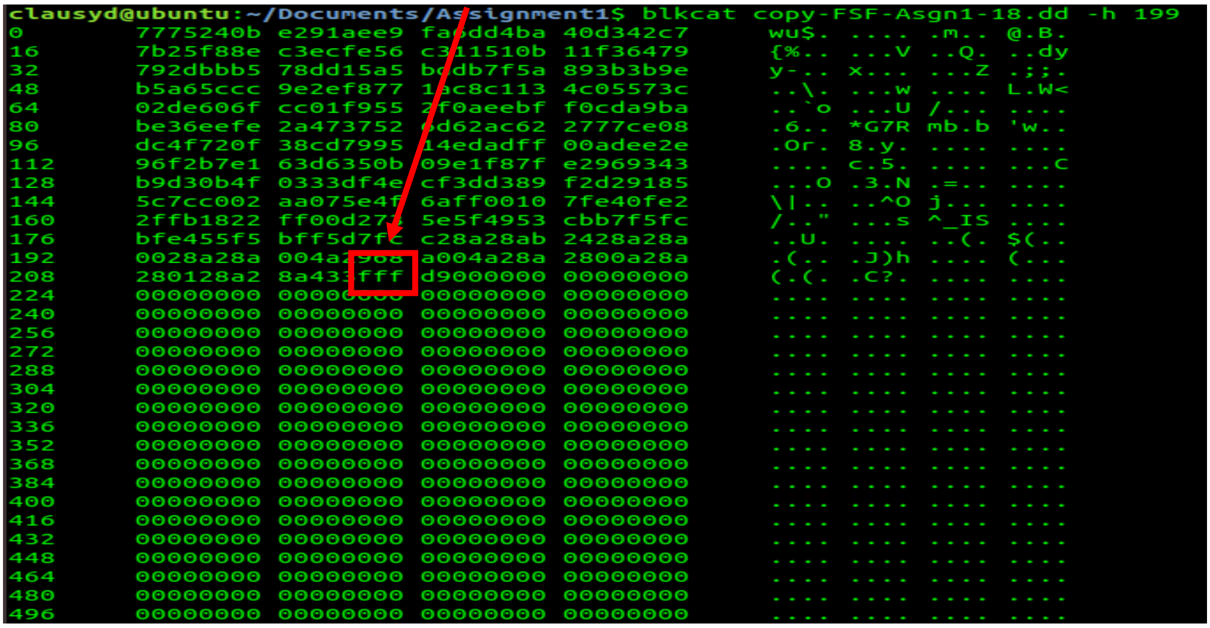



*Figure 13: Root Directory Fat12 Layout*

Because of our data struture shown in figure 11 we no the Mean.zip run from 35-199 this maens that there should be three FFF's found at sector 199 . If you refer to figure 14 you will see the three FFF's highlighted in red.



```
clausyd@ubuntu:~/Documents/Assignment1$ blkcat copy-FSF-Asgn1-18.dd -h 199
0       7775240b e291aee9 fa6dd4ba 40d342c7    wu$. .... .m.. @.B.
16      7b25f88e c3ecfe56 c311510b 11f36479    {%.. ...V ..Q. ..dy
32      792dbbb5 78dd15a5 bddb7f5a 893b3b9e    y-.. x... ...Z .;;.
48      b5a65ccc 9e2ef877 1ac8c113 4c05573c    ..\. ...w .... L.W<
64      02de606f cc01f955 2f0aeebf f0cda9ba    ..`o ...U /... ....
80      be36eefe 2a473752 6d62ac62 2777ce08    .6.. *G7R mb.b 'w..
96      dc4f720f 38cd7995 14edadff 00adee2e    .Or. 8.y. .... ....
112     96f2b7e1 63d6350b 09e1f87f e2969343    .... c.5. .... ...C
128     b9d30b4f 0333df4e cf3dd389 f2d29185    ...O .3.N .=.. ....
144     5c7cc002 aa075e4f 6aff0010 7fe40fe2    \|.. ..^O j... ....
160     2ffb1822 ff00d273 5e5f4953 cbb7f5fc    /.." ...s ^_IS ....
176     bfe455f5 bff5d7fc c28a28ab 2428a28a    ..U. .... ..(. $(..
192     0028a28a 004a2968 a004a28a 2800a28a    .(.. .J)h .... (...
208     280128a2 8a433fff d9000000 00000000    (.(. .C?. .... ....
224     00000000 00000000 00000000 00000000    .... .... .... ....
240     00000000 00000000 00000000 00000000    .... .... .... ....
256     00000000 00000000 00000000 00000000    .... .... .... ....
272     00000000 00000000 00000000 00000000    .... .... .... ....
288     00000000 00000000 00000000 00000000    .... .... .... ....
304     00000000 00000000 00000000 00000000    .... .... .... ....
320     00000000 00000000 00000000 00000000    .... .... .... ....
336     00000000 00000000 00000000 00000000    .... .... .... ....
352     00000000 00000000 00000000 00000000    .... .... .... ....
368     00000000 00000000 00000000 00000000    .... .... .... ....
384     00000000 00000000 00000000 00000000    .... .... .... ....
400     00000000 00000000 00000000 00000000    .... .... .... ....
416     00000000 00000000 00000000 00000000    .... .... .... ....
432     00000000 00000000 00000000 00000000    .... .... .... ....
448     00000000 00000000 00000000 00000000    .... .... .... ....
464     00000000 00000000 00000000 00000000    .... .... .... ....
480     00000000 00000000 00000000 00000000    .... .... .... ....
496     00000000 00000000 00000000 00000000    .... .... .... ....
```

*Figure 14: End of File*

If you refer to figure 15 shown is the Fat tables. The Fat tables are found in sector 1 and 2. What is highlighted in red is the end of file markers. What you will see is values:

0aa7800a fff = a0 0a7 08a fff

0a7 = 167

08a = 168

```
clausyd@ubuntu:~/Documents/Assignment1$ blkcat copy-FSF-Asgn1-18.dd -h 1
0       f0ffff00 00000560 00078000 09a0000b    .... ...` .... ....
16      c0000de0 000f0001 11200113 40011560    .... .... . .. @..`
32      01178001 19a0011b c0011de0 011f0002    .... .... .... ....
48      21200223 40022560 02278002 29a0022b    ! .# @.%` .'.. )..+
64      c0022de0 022f0003 31200333 40033560    ..-. ./.. 1 .3 @.5`
80      03378003 39a0033b c0033de0 033f0004    .7.. 9..; ..=. .?..
96      41200443 40044560 04478004 49a0044b    A .C @.E` .G.. I..K
112     c0044de0 044f0005 51200553 40055560    ..M. .O.. Q .S @.U`
128     05578005 59a0055b c0055de0 055f0006    .W.. Y..[ ..]. ._..
144     61200663 40066560 06678006 69a0066b    a .c @.e` .g.. i..k
160     c0066de0 066f0007 71200773 40077560    ..m. .o.. q .s @.u`
176     07778007 79a0077b c0077de0 077f0008    .w.. y..{ ..}. ....
192     81200883 40088560 08878008 89a0088b    . .. @..` .... ....
208     c0088de0 088f0009 91200993 40099560    .... .... . .. @..`
224     09978009 99a0099b c0099de0 099f000a    .... .... .... ....
240     a1200aa3 400aa560 0aa7800a fffffab    . .. @..` .... ....
256     c00aade0 0aaf000b b1200bb3 400bb560    .... .... . .. @..`
272     0bb7800b b9a00bbb c00bbde0 0bbf000c    .... .... .... ....
288     c1200cc3 400cc560 0cc7800c c9a00ccb    . .. @..` .... ....
304     c00ccde0 0ccf000d d1200dd3 400dd560    .... .... . .. @..`
320     0dd7800d d9a00ddb c00ddde0 0ddf000e    .... .... .... ....
336     e1200ee3 400ee560 0ee7800e e9a00eeb    . .. @..` .... ....
352     c00eede0 0eef000f f1200ff3 400ff560    .... .... . .. @..`
368     0ff7800f f9a00ffb c00ffde0 0fff0010    .... .... .... ....
384     01211003 41100561 10078110 09a1100b    .!.. A..a .... ....
400     c1100de1 100f0111 11211113 41111561    .... .... .!.. A..a
416     11178111 19a1111b c1111de1 111f0112    .... .... .... ....
432     21211223 41122561 12278112 29a1122b    !!.# A.%a .'.. )..+
448     c1122de1 122f0113 31211333 41133561    ..-. ./.. 1!.3 A.5a
464     13378113 39a1133b c1133de1 133f0114    .7.. 9..; ..=. .?..
480     41211443 41144561 14478114 49a1144b    A!.C A.Ea .G.. I..K
496     c1144de1 144f0115 51211553 41155561    ..M. .O.. Q!.S A.Ua
```



*Figure 15: Fat Tables*

Fat12 file systems are laid out different from other Fat file systems. “Since 12 bits is not an integral number of bytes, we have to specify how these are arranged. Two FAT12 entries are stored into three bytes; if these bytes are uv,wx,yz then the entries are xuv and yzw” (www.win.tue.nl, 2019). Also, with the use of little Indian means that values are reversed. The values above represent 167 and 168 and then the end of the file. Because of the offset this would make since in that you add 32 to 167 and you get 199 which is the end of the MEANS.zip file.

## 5. Storage/Deletion Process of a File

The deletion process is similar to the creation process except when it finds the value in the Fat tables it places a 0 in there. When a user deletes a file the boot sector will process through the root directory until it finds the file the user is deleting. It will look for the corresponding cluster number shown in figure 16 below this number is 90. It will then process through the cluster until it finds cluster 90 and will look at the next cluster value found, and, in this case, it clusters 200. Next it will process through the clusters again until it finds 200 and it finds the contents of the files. Once the boot sector has located the cluster it then reviews the Fat tables until it finds 200 and write a zero into this slot. After this the file will appear as unallocated because the starting value will be a zero.

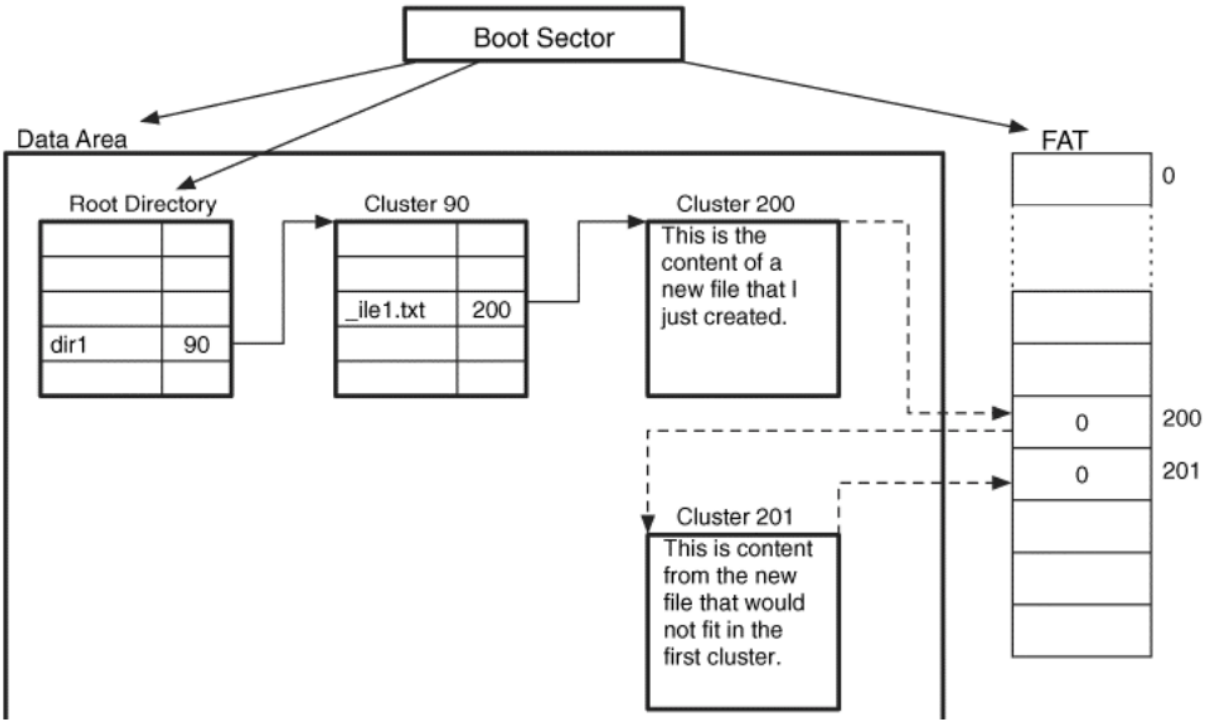



*Figure 16: Cluster chain deletion process.*

# 6. Recovered Files

In order to identify the files contained on the image the investigator ran the following command from the SleuthKit tool:

**fls copy-FSF-Asgn1-18.dd**

This command returned the file name and locations as shown in figure 17. There were three files found. New folder was at location 4, Means.zip was found at location 6 and Secret was at location 8.

- New folder
- Means.zip
- Secret

```
clausyd@ubuntu:~/Documents/Assignment1$ fls copy-FSF-Asgn1-18.dd
d/d * 4:        New folder
r/r 6:  Means.zip
d/d 8:  Secret
v/v 45779:      $MBR
v/v 45780:      $FAT1
v/v 45781:      $FAT2
d/d 45782:      $OrphanFiles
```

*Figure 17: Files contained within the image.*

## 6.1 A Deleted File

### 6.1.1 _EWFOL~1

The first step is to investigate the _EWFOL~1 file in greater detail. To do that the istat command was used and this gives us more detail about the directory entry at location 4. If you refer to figure 18 you can see that it tells us a size of the file which is 512 this is consistent with the number of sectors. It gives us a file name as well.

**istat copy-FSF-Asgn1-18.dd 4**

```
clausyd@ubuntu:~/Documents/Assignment1$ istat copy-FSF-Asgn1-18.dd 4
Directory Entry: 4
Not Allocated
File Attributes: Directory
Size: 512
Name: _EWFOL~1

Directory Entry Times:
Written:        2018-10-16 13:24:20 (IST)
Accessed:       0000-00-00 00:00:00 (UTC)
Created:        0000-00-00 00:00:00 (UTC)

Sectors:
33
```

*Figure 18: Using the icat command to gain more information on _EWFOL~1.*

Now that we now the sector it enables us to view the hexadecimal values of a sector using the blkcat command as seen below. This istat command stated that there is something at sector 33 so that is the first place in investigate. If you refer to figure 19 below that there was something in that sector but not too much.

**blkcat copy-FSF-Asgn1-18.dd -h 33**

```
clausyd@ubuntu:~/Documents/Assignment1$ blkcat copy-FSF-Asgn1-18.dd -h 33
0       2e202020 20202020 20202010 00000000     .          . ....
16      00000000 00000a6b 504d0200 00000000     .... ...k PM.. ....
32      2e2e2020 20202020 20202010 00000000     ..         . ....
48      00000000 00000a6b 504d0000 00000000     .... ...k PM.. ....
64      e54e006f 00740065 002e000f 00867400     .N.o .t.e .... ..t.
80      78007400 2e007200 74000000 66000000     x.t. ..r. t... f...
96      e54f5445 7e312020 52544600 105d036a     .OTE ~1   RTF. .].j
112     504d0000 0000046a 504d0300 e9000000     PM.. ...j PM.. ....
128     00000000 00000000 00000000 00000000     .... .... .... ....
144     00000000 00000000 00000000 00000000     .... .... .... ....
160     00000000 00000000 00000000 00000000     .... .... .... ....
176     00000000 00000000 00000000 00000000     .... .... .... ....
192     00000000 00000000 00000000 00000000     .... .... .... ....
208     00000000 00000000 00000000 00000000     .... .... .... ....
224     00000000 00000000 00000000 00000000     .... .... .... ....
240     00000000 00000000 00000000 00000000     .... .... .... ....
256     00000000 00000000 00000000 00000000     .... .... .... ....
272     00000000 00000000 00000000 00000000     .... .... .... ....
288     00000000 00000000 00000000 00000000     .... .... .... ....
304     00000000 00000000 00000000 00000000     .... .... .... ....
320     00000000 00000000 00000000 00000000     .... .... .... ....
336     00000000 00000000 00000000 00000000     .... .... .... ....
352     00000000 00000000 00000000 00000000     .... .... .... ....
368     00000000 00000000 00000000 00000000     .... .... .... ....
384     00000000 00000000 00000000 00000000     .... .... .... ....
400     00000000 00000000 00000000 00000000     .... .... .... ....
416     00000000 00000000 00000000 00000000     .... .... .... ....
432     00000000 00000000 00000000 00000000     .... .... .... ....
448     00000000 00000000 00000000 00000000     .... .... .... ....
464     00000000 00000000 00000000 00000000     .... .... .... ....
480     00000000 00000000 00000000 00000000     .... .... .... ....
496     00000000 00000000 00000000 00000000     .... .... .... ....
```

*Figure 19: Hexadecimal value for sector 33.*

The next step is to look in the next sector again utilizing or blkcat extractor. The following command was issued.

**blkcat copy-FSF-Asgn1-18.dd -h 34**

If you refer to figure 20 you will see some interesting information. We know that there is a file at this location. It tells use the file type which is rich text format highlighted in the red.  The file looks to end between 224-240.

```
clausyd@ubuntu:~/Documents/Assignment1$ blkcat copy-FSF-Asgn1-18.dd -h 34
0       7b5c7274 66315c61 6e73695c 616e7369     {\rt f1\a nsi\ ansi
16      63706731 3235325c 64656666 305c6e6f     cpg1 252\ deff 0\no
32      7569636f 6d706174 5c646566 6c616e67     uico mpat \def lang
48      32313038 7b5c666f 6e747462 6c7b5c66     2108 {\fo nttb l{\f
64      305c666e 696c5c66 63686172 73657430     0\fn il\f char set0
80      2043616c 69627269 3b7d7d0d 0a7b5c2a      Cal ibri ;}}. .{\*
96      5c67656e 65726174 6f722052 69636865     \gen erat or R iche
112     64323020 31302e30 2e313632 39397d5c     d20  10.0 .162 99}\
128     76696577 6b696e64 345c7563 31200d0a     view kind 4\uc 1 ..
144     5c706172 645c7361 3230305c 736c3237     \par d\sa 200\ sl27
160     365c736c 6d756c74 315c6630 5c667332     6\sl mult 1\f0 \fs2
176     325c6c61 6e673630 20492073 61772074     2\la ng60  I s aw t
192     68657365 206f6e6c 696e652c 20776861     hese  onl ine,  wha
208     7420646f 20796f75 20746869 6e6b3f5c     t do  you  thi nk?\
224     7061720d 0a7d0d0a 00000000 00000000     par. .}.. .... ....
240     00000000 00000000 00000000 00000000
```

*Figure 20: Hexadecimal value for sector 34*

The SleuthKit enables investigators to see deleted files through the following command.

**fls -r copy-FSF-Asgn1-18.dd**

If you refer to figure 21 you will notice that there now an extra file called Note.txt.rtf this is a file which had been deleted from the New Folder. Now that we varied there was a file deleted, we can extract the file.

```
clausyd@ubuntu:~/Documents/Assignment1$ fls -r copy-FSF-Asgn1-18.dd
d/d * 4:        New folder
+ r/r * 230:    Note.txt.rtf
r/r 6:  Means.zip
d/d 8:  Secret
v/v 45779:      $MBR
v/v 45780:      $FAT1
v/v 45781:      $FAT2
d/d 45782:      $OrphanFiles
```

*Figure 21: fls command to see deleted files.*

The file was extracted using the following command:

**icat copy-FSF-Asgn1-18.dd 230 > Note.txt.rtf**

The command shown above is the icat command it states the image that we want to extract from then it stated the directory entry and points to the name of the deleted file. If you refer to figure 22 you will see that it was successful as the directory was listed using the ls command and you will notice the file highlighted in red is the file that was just extracted.

```
clausyd@ubuntu:~/Documents/Assignment1$ icat copy-FSF-Asgn1-18.dd 230 > Note.txt
.rtf
clausyd@ubuntu:~/Documents/Assignment1$ ls
CF-Asgn1-19.dd.zip     MEANS.jpg   Note.txt.rtf
copy-FSF-Asgn1-18.dd   Means.zip   SECRET
clausyd@ubuntu:~/Documents/Assignment1$
```

*Figure 22: Deleted file extracted from file system.*

The next step the investigator took was to take a hash values of the extracted file. This is so the file can be verified if wanted in a court of law. If you refer to figure 23 you will see the generated hash values and the commands that were issued to generate them.

```
clausyd@ubuntu:~/Documents/Assignment1$ md5sum Note.txt.rtf
45ac853626eb3b96fdb7d539e0fbb0c0  Note.txt.rtf
clausyd@ubuntu:~/Documents/Assignment1$ shasum Note.txt.rtf
1563cfa0f7933d25714d9fedfa82fe1268821f38  Note.txt.rtf
clausyd@ubuntu:~/Documents/Assignment1$ sha256sum Note.txt.rtf
9ed8bb3d49301c9ccbdc2995bf85f552ef1f54f9e918c3d848aa97d7d7e91f09  Note.txt.rtf
clausyd@ubuntu:~/Documents/Assignment1$
```

*Figure 23: Hash values of Note.txt.rtf.*

As stated in section 1, there is a second tool going to be used to verify the files that are found. If you refer to figure 24 below you will see conformation of the New Folder file and the Note.txt.rtf file. If you look to the left of the image you will see the evidence tree containing the image and the root directory and unallocated space. You will notice that the New folder in a darker shade that is because it is selected. Once selected you can see the Note.txt.rtf file contained inside that folder and it is marked with an x meaning it's a deleted file. Underneath that you can see the hexadecimal values for that file. This confirms the investigation undertaken with the SleuthKit.

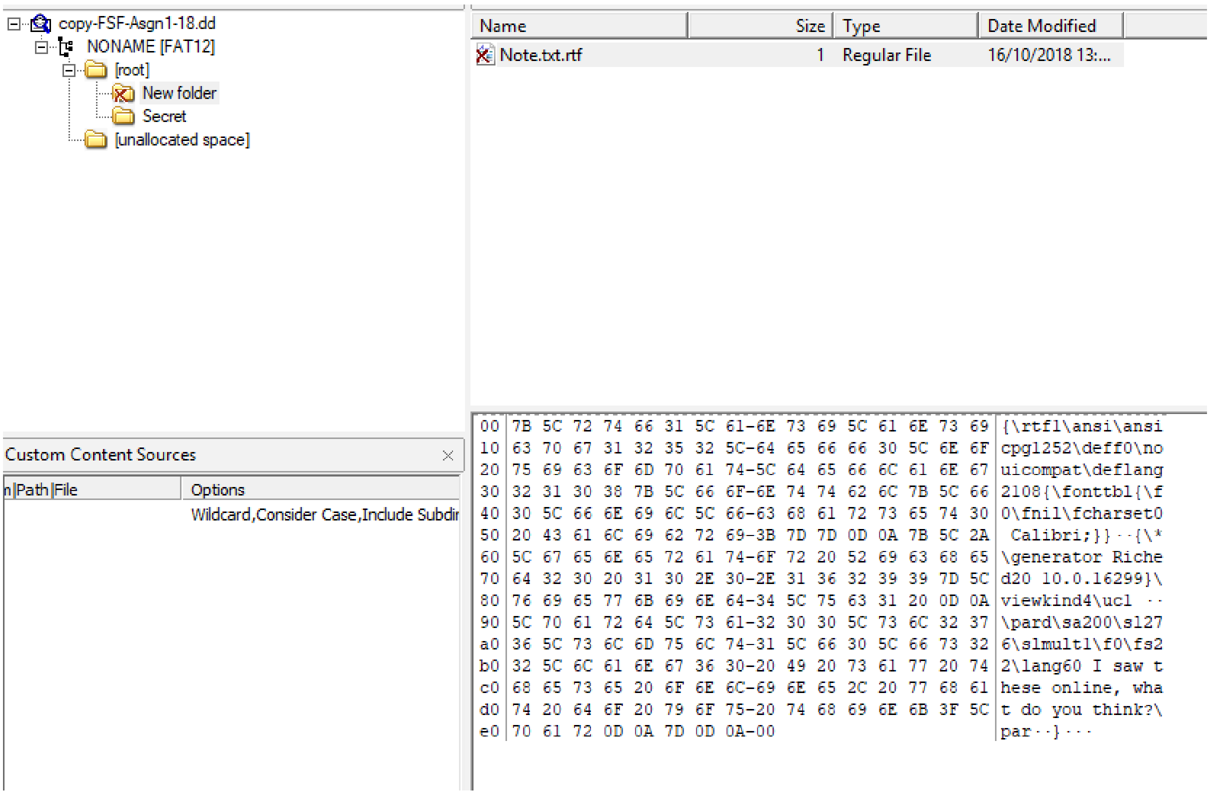



*Figure 24: FTK imager verifying New Folder file and the Note.txt.rtf file*

If you refer to figure 25 you will see the extracted file opened in the text editor.

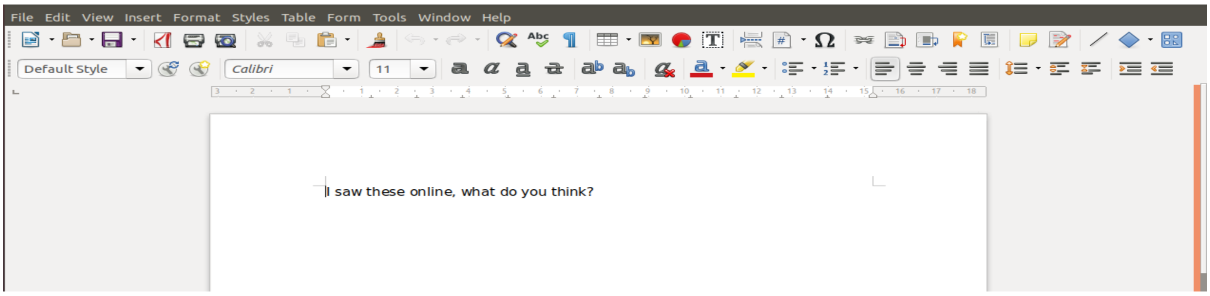



*Figure 25: Deleted file open in text editor.*

## 6.2 A File with a Mismatched Extension

A file with a mismatch extension is an extension that don't match the file type. This may be a .jpg file type represented as a zip file like seen in this investigation. This may be done to hide the contents of a file or to make investigators think the file is something different then is actually is.

### 6.2.1 Means.zip

After performing fls command the following file was seen as a .zip file extension. The first step to investigate this file further with the following command:

**istat copy-FSF-Asgn1-18.dd 6**

The investigator used the istat command on the copied disk image and specifying location 6 as seen in figure 26. As you can see this gives far more detail on the file. The first attribute that was looked at was the file size. The investigator could see that the file size was 84185. In order to see if this is an accurate measurement the investigator can calculate the number of sectors and multiply the total by 512 which is the size of one sector. In total there is 165 sectors. 165*512=84480 this number is fairly close to the size giving below. Not too much irregularity on the file size.

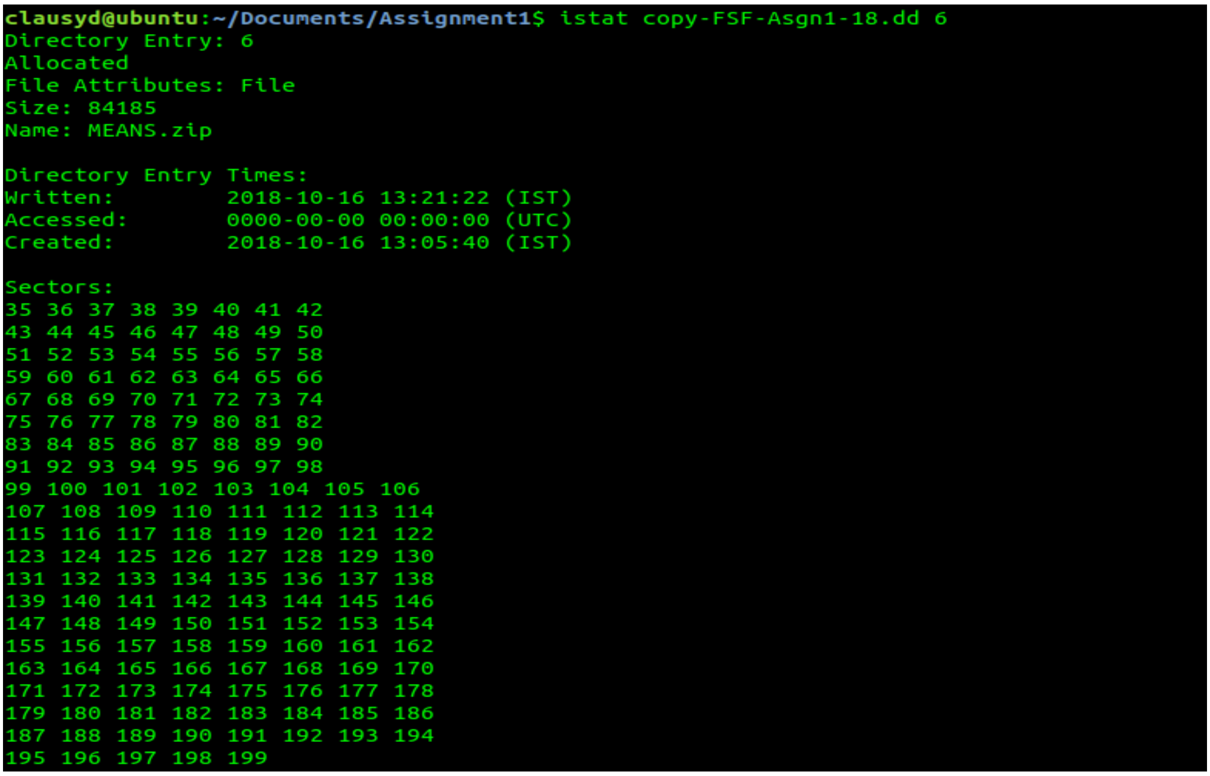



*Figure 26: Using the istat command to get more detail on the Means.zip file.*

The next investigation step for this file was to us the icat command. The icat command will output the file to the current working directory. The command used states the image that the file is contained in the location of the file which in the case is 6 and the pointer to the file name as shown below.

**icat copy-FSF-Asgn1-18.dd 6 > Means.zip**

If you refer to figure 27 you will see the command being used. The investigator then used the Linux command ls to list the current directory where you will see the file. The investigator then tried to unzip the file and was presented with the following error seen in figure 21.

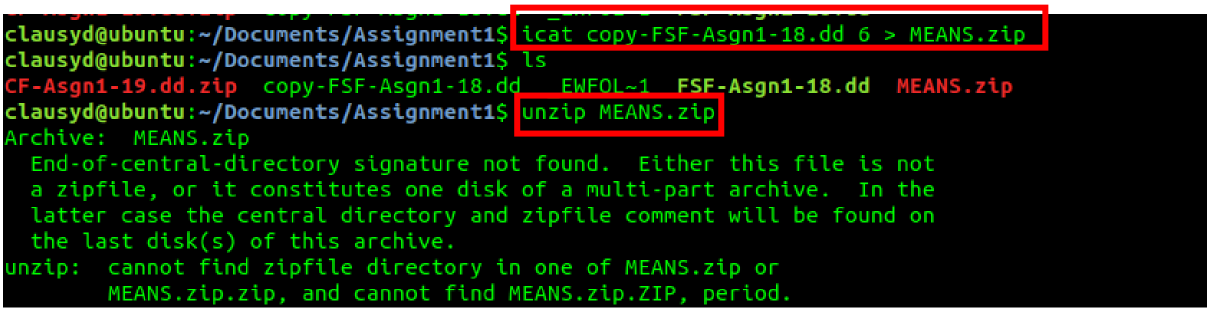



*Figure 27: Outputting Means.zip to the current working directory.*

The file required further investigating to locate the file type and the signature was not found. To do this there was another command the can be used to investigate the hexadecimal value of the file where you can locate the file signature. The command used was:

**blkcat copy-FSF-Asgn1-18.dd -h 35**

After using the istat command the investigator knew that the first sector was 35 and the first sector is normally where you locate the file signature. If you refer to figure 28 the part highlighted in red the first six digits is the file signature and after checking this value against the file signature database it was a match for a .jpeg file extension. Also highlighted in red you will see the asci value is also shown.

```
clausyd@ubuntu:~/Documents/Assignment1$ blkcat copy-FSF-Asgn1-18.dd -h 35
0       ffd8ffe0 00104a46 49460001 01010060     .... ..JF IF.. ...`
16      00600000 ffe1111a 45786966 00004d4d     .`.. .... Exif ..MM
32      002a0000 00080005 013b0002 00000006     .*.. .... .;.. ....
48      00000856 87690004 00000001 0000085c     ...V .i.. .... ...\
64      9c9d0001 0000000c 000010d4 9c9e0001     .... .... .... ....
80      00000032 000010e0 ea1c0007 0000080c     ...2 .... .... ....
96      0000004a 00000000 1cea0000 00080000     ...J .... .... ....
112     00000000 00000000 00000000 00000000     .... .... .... ....
```

*Figure 28: Investigating sectors with blk.*

If you refer to figure 29 you will see another Linux command just to confirm the file type. It clearly states that the file type is a jpg.

```
clausyd@ubuntu:~/Documents/Assignment1$ file MEANS.zip
MEANS.zip: JPEG image data, JFIF standard 1.01, resolution (DPI), density 96x96, segment length 16, Exif S
tandard: [TIFF image data, big-endian, direntries=5], baseline, precision 8, 869x537, frames 3
```

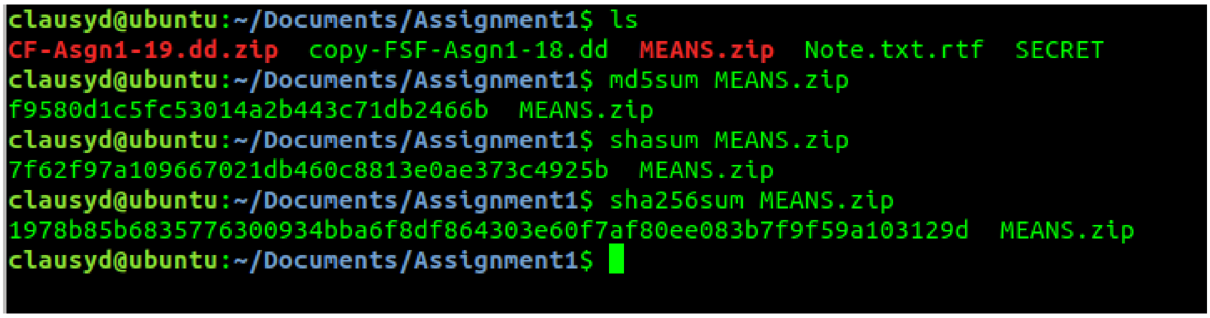
*Figure 29: Linux file command.*

While investigating the rest of the sector using the blkcat command this was some text encoded within the file. Found in sector 45 there was some text saying this is how we can do it. This was also confirmed with the FTK imager as shown in figure 35 below.

```
clausyd@ubuntu:~/Documents/Assignment1$ blkcat copy-FSF-Asgn1-18.dd -h 45
0       3c2f6463 3a637265 61746f72 3e3c6463     </dc :cre ator ><dc
16      3a737562 6a656374 3e3c7264 663a4261     :sub ject ><rd f:Ba
32      6720786d 6c6e733a 7264663d 22687474     g xm lns: rdf= "htt
48      703a2f2f 7777772e 77332e6f 72672f31     p:// www. w3.o rg/1
64      3939392f 30322f32 322d7264 662d7379     999/ 02/2 2-rd f-sy
80      6e746178 2d6e7323 223e3c72 64663a6c     ntax -ns# "><r df:l
96      693e5468 69732069 7320686f 77207765     i>Th is i s ho w we
112     2063616e 20646f20 69743c2f 7264663a      can  do  it</ rdf:
128     6c693e3c 2f726466 3a426167 3e0d0a09     li>< /rdf :Bag >...
144     09093c2f 64633a73 75626a65 63743e3c     ..</ dc:s ubje ct><
160     2f726466 3a446573 63726970 74696f6e     /rdf :Des crip tion
176     3e3c7264 663a4465 73637269 7074696f     ><rd f:De scri ptio
192     6e207264 663a6162 6f75743d 22757569     n rd f:ab out= "uui
208     643a6661 66356264 64352d62 6133642d     d:fa f5bd d5-b a3d-
224     31316461 2d616433 312d6433 33643735     11da -ad3 1-d3 3d75
240     31383266 31622220 786d6c6e 733a4d69     182f 1b"  xmln s:Mi
256     63726f73 6f667450 686f746f 3d226874     cros oftP hoto ="ht
272     74703a2f 2f6e732e 6d696372 6f736f66     tp:/ /ns. micr osof
```

*Figure 30: Looking for more details in the file.*

Once the file type was identified and before the image was open some hash values needed to be generated in case the file is need in a court of law. This way a jury will be able to verify the file have not been compromised in any way since the investigation was done. If you refer to figure 31 you will see the commands that were used, and the hash values generated for the jpg image.



```
clausyd@ubuntu:~/Documents/Assignment1$ ls
CF-Asgn1-19.dd.zip  copy-FSF-Asgn1-18.dd  MEANS.zip  Note.txt.rtf  SECRET
clausyd@ubuntu:~/Documents/Assignment1$ md5sum MEANS.zip
f9580d1c5fc53014a2b443c71db2466b  MEANS.zip
clausyd@ubuntu:~/Documents/Assignment1$ shasum MEANS.zip
7f62f97a109667021db460c8813e0ae373c4925b  MEANS.zip
clausyd@ubuntu:~/Documents/Assignment1$ sha256sum MEANS.zip
1978b85b6835776300934bba6f8df864303e60f7af80ee083b7f9f59a103129d  MEANS.zip
clausyd@ubuntu:~/Documents/Assignment1$
```

*Figure 31: Hash values generated for MEANS.zip.jpg image.*

The next step the investigator took was to rename the file to a .jpg as seen in figure 32. Once the file was renamed the hash values were generated again. This proves the file was not altered in any way. If you refer to figure 31 and compare with the hash value sin figure 33 you will see that they match.

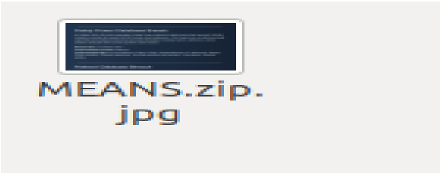
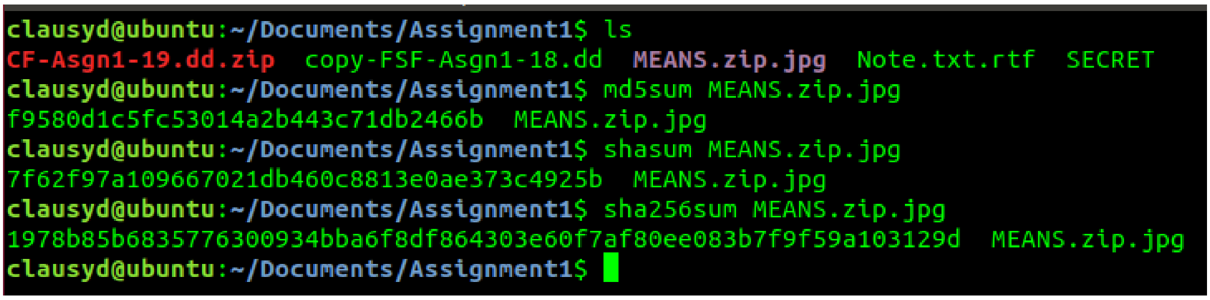



*Figure 32: Renaming the file to .jpg.*



```
clausyd@ubuntu:~/Documents/Assignment1$ ls
CF-Asgn1-19.dd.zip  copy-FSF-Asgn1-18.dd  MEANS.zip.jpg  Note.txt.rtf  SECRET
clausyd@ubuntu:~/Documents/Assignment1$ md5sum MEANS.zip.jpg
f9580d1c5fc53014a2b443c71db2466b  MEANS.zip.jpg
clausyd@ubuntu:~/Documents/Assignment1$ shasum MEANS.zip.jpg
7f62f97a109667021db460c8813e0ae373c4925b  MEANS.zip.jpg
clausyd@ubuntu:~/Documents/Assignment1$ sha256sum MEANS.zip.jpg
1978b85b6835776300934bba6f8df864303e60f7af80ee083b7f9f59a103129d  MEANS.zip.jpg
clausyd@ubuntu:~/Documents/Assignment1$
```

*Figure 33: Regenerated hash values after file renaming.*

Once this was done you will see that the image could then be open as seen in figure 34. This was clearly a file mismatch or a way or throwing investigators of the scient in their investigation.

**Compromised data:** Email addresses, IP addresses, Passwords, Usernames

Paddy Power Database Breach

In October 2010, the Irish bookmaker Paddy Power suffered a data breach that exposed 750,000 customer records with nearly 600,000 unique email addresses. The breach was not disclosed until July 2014 and contained extensive personal information including names, addresses, phone numbers and plain text security questions and answers.

**Breach date:** 25 October 2010
**Compromised accounts:** 590,954
**Compromised data:** Account balances, Dates of birth, Email addresses, IP addresses, Names, Phone numbers, Physical addresses, Security questions and answers, Usernames, Website activity

Patreon Database Breach

In October 2015, the crowdfunding site Patreon was hacked and over 16GB of data was released publicly. The dump included almost 14GB of database records with more than 2.3M unique email

*Figure 34: .jpg that was outputted to the working directory.*

If you refer to figure 35 you will see the FTK imager verifying the file. In the image it is showing as a .zip file but it you look to the hexadecimal code at the bottom you will see the hexadecimal value for the file extension FFD8FF and also you can see the asci value JFIF. This proves the file extension mismatch.

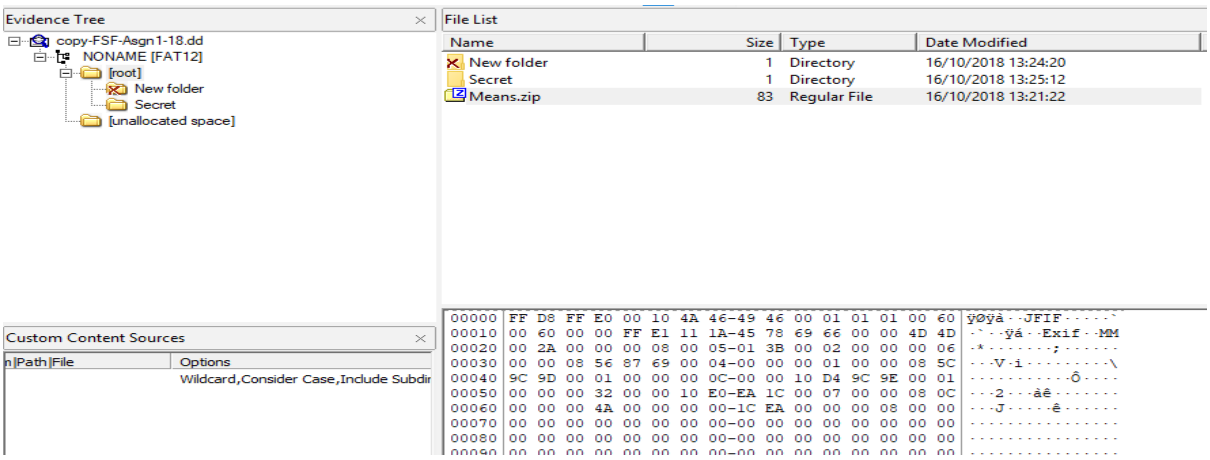


Figure 35: FTK imager verifying MEANS.zip file.

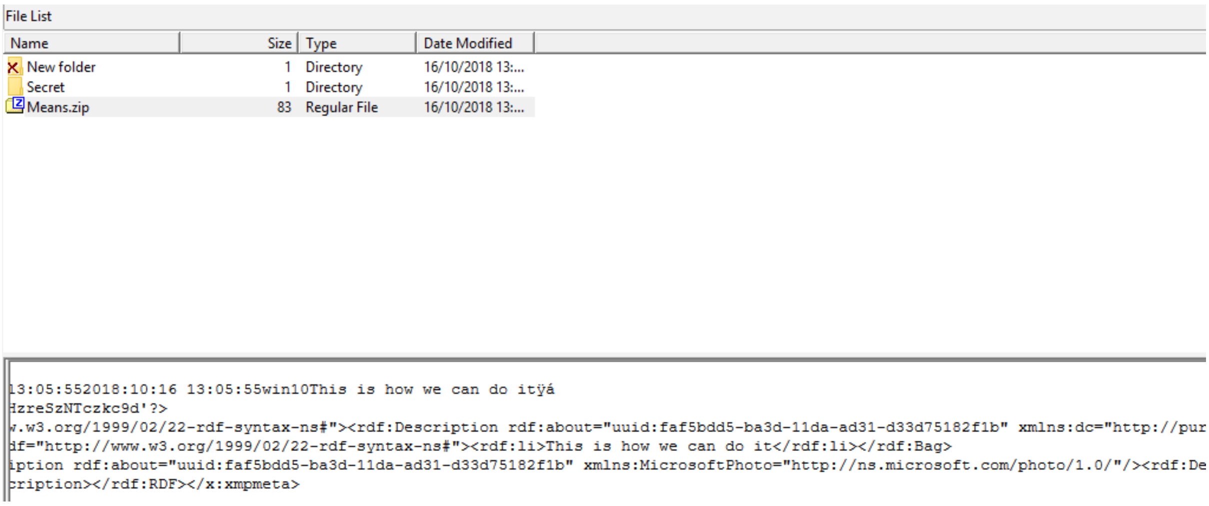


Figure 36: Showing encoded text using FTK imager.

## 6.3 An Allocated File

### 6.3.1 SECRET

The next file to investigate was Secret file located at 8. To gain more detail about the file the istat command was used seen below:

**istat copy-FSF-Asgn1-18.dd 8**

If you refer to figure 37 you will see some detail about the file in question. You will notice the attributes of the file is shown as a directory and hidden. There is no file size and no sectors shown so it was hard to determine what the file is or what size it was.

```
clausyd@ubuntu:~/Documents/Assignment1$ istat copy-FSF-Asgn1-18.dd 8
Directory Entry: 8
Allocated
File Attributes: Directory, Hidden
Size: 0
Name: SECRET

Directory Entry Times:
Written:        2018-10-16 13:25:12 (IST)
Accessed:       0000-00-00 00:00:00 (UTC)
Created:        0000-00-00 00:00:00 (UTC)

Sectors:
```

*Figure 37: Using icat command to gain more information on Secret file.*

Being able to reference back to our mapped our data structure and because we knew the number of sectors used by the MEANS.zip file which was 165. The investigator was able to map the MEANS.zip file to the section of the data structure that was the same size as file size with the arrow pointing to it. Then there were only two files left. One file contains only one sector which was 512 bytes and the other file that contains 252 sectors. Using the istat command we could see that the file called _EWFOL~1 only contained one sector.

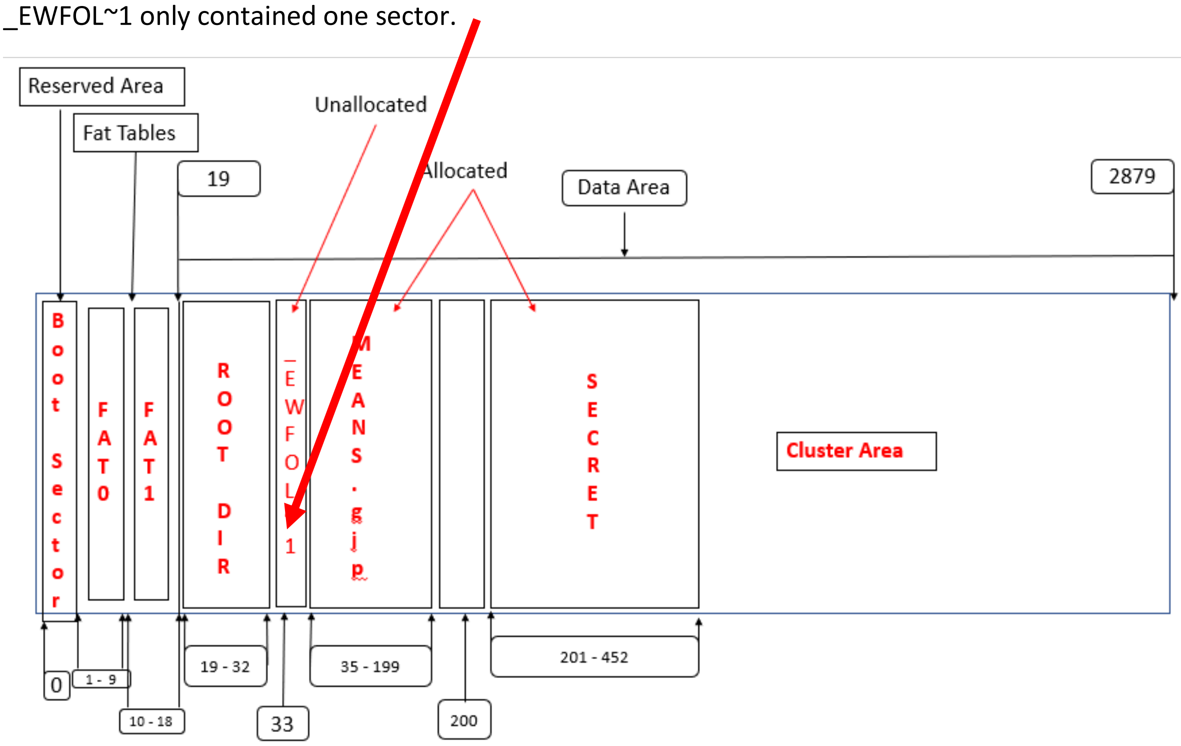



*Figure 38: Matching the files to the sections of the data structure.*

The first command that was used was fsstat which showed the layout of the whole file system. In particular it showed the fat contents as seen in figure 39. This showed that there was something contained between sector 201-252.

```
FAT CONTENTS (in sectors)
-----------------------------------
35-199 (165) -> EOF
200-200 (1) -> EOF
201-452 (252) -> EOF
```

*Figure 39: FAT CONTENTS.*

By using the dd command investigators can carve out sections of the FAT32 file system. The section this investigation was interested in was 201-452. If you look at the command below it states the image to carve, it then states the file that you want carved out. It gives the size of each sector and then the carving starts. In this case it was 201 and then count out 254 this was a mistake it should be 252 because 251+201=252 but it didn't seem to matter adding a couple of extra sectors if you refer to figure 40 you will see that it was successful.

**dd if=copy-FSF-Asgn1-18.dd of=Secret bs=512 skip=201 count=254**

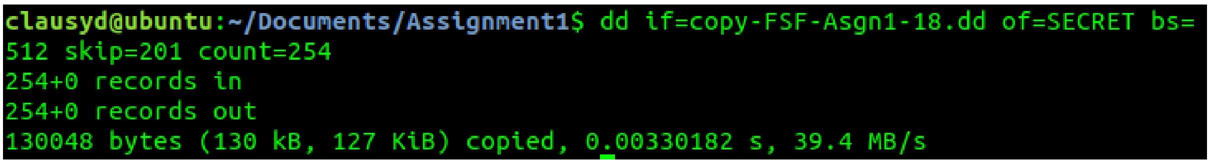



*Figure 40: Carving out section of the file system.*

Once the file is carved out the next step is to generate hash values of the file. This is so the file can be verified in court and to make sure the file hasn't been compromised in any way. If you refer to figure 35 you will see the command used and hash values that were generated.

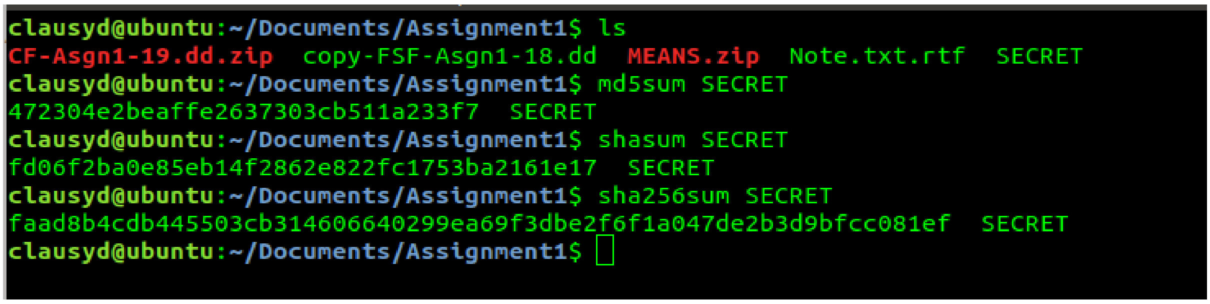



*Figure 41: Hash values generated for SECRET.png.*

If you refer to figure 42 you will see the image in the working directory.

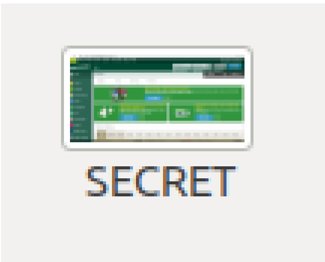



*Figure 42: Image carved out to working directory.*

If you refer to figure 43 below you will see the recovered image.

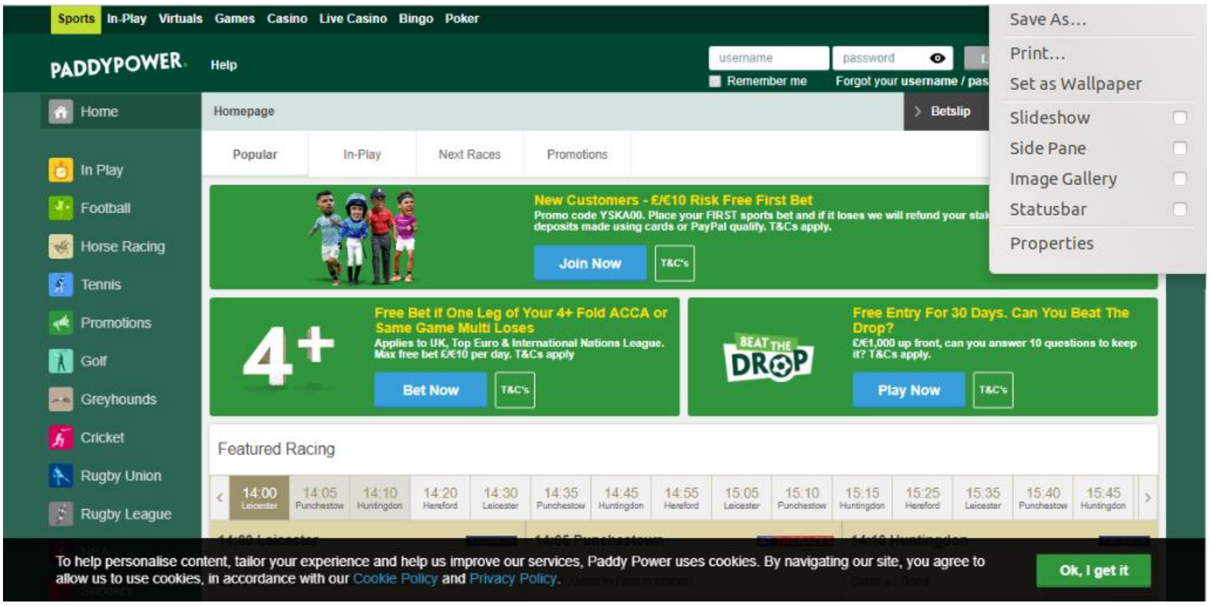



*Figure 43: Image found after using file carving command.*

If you refer to figure 44 you will see the FTK tool being used. You will notice the Secret file highlighted in the evidence tree on the left and contain with the folder you can see the png image that was carved out of that file.

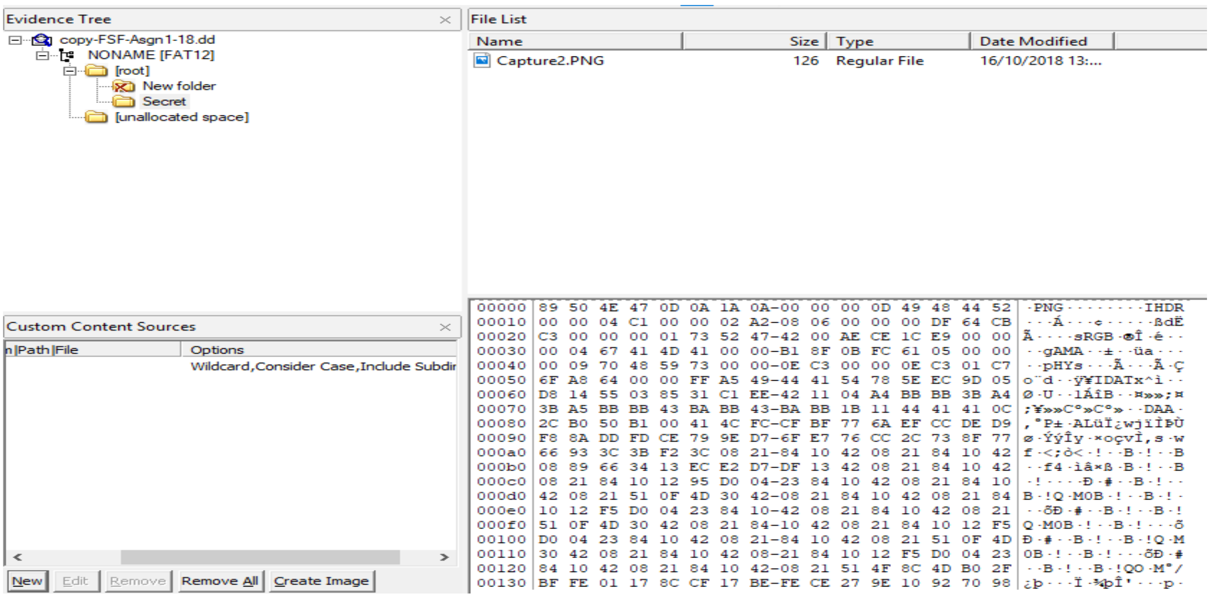



*Figure 44: FTK imager verifying Secret file.*

# 7. Other Commands User in Investigation

There is a command called string for Linux distros that will return any sequence of strings for on the image as referred to in figure 45. If you refer to figure 46 below there is some interesting detail in this. It lists all the file and directory name. You will notice the RTF file has some interesting links pointing back to schemas on w3.org which is the world wide web consortium. This file is using Resource description framework to point to the images. And Dublin DC is also being used, these are XML standards used for encoding metadata.

**strings copy-FSF-Asgn1-18.dd**

```
clausyd@ubuntu:~/Documents/Assignment1$ strings copy-FSF-Asgn1-18.dd
WINIMAGE
P            FAT12
Cannot load from harddisk.
Insert Systemdisk and press any key.
Disk formatted with WinImage 6.50 (c) 1993-2004 Gilles Vollant
see http://www.winimage.com
Bootsector from C.H. Hochstatter
No Systemdisk. Booting from harddisk.
EWFOL~1
MEANS   ZIP
SECRET
&kPM
.
..
OTE~1  RTF
```

*Figure 45: Strings command.*

Highlighted is red in figure 46 there is two sentences.

- I saw these online, what do you think?
- This is how we can do it.

The first sentence is referencing the Means.zip folder where the data breach was identified. The second sentence is then referencing the Paddy Power login page where they could use the unique email address to gain access to user accounts.

```
{\*\generator Riched20 10.0.16299}\viewkind4\uc1
\pard\sa200\sl276\slmult1\f0\fs22\lang60 I saw these online, what do you think?\par
JFIF
Exif
win10
2018:10:16 13:05:55
2018:10:16 13:05:55
whttp://ns.adobe.com/xap/1.0/
<?xpacket begin='
' id='W5M0MpCehiHzreSzNTczkc9d'?>
<x:xmpmeta xmlns:x="adobe:ns:meta/"><rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"><r
df:Description rdf:about="uuid:faf5bdd5-ba3d-11da-ad31-d33d75182f1b" xmlns:dc="http://purl.org/dc/eleme
nts/1.1/"/><rdf:Description rdf:about="uuid:faf5bdd5-ba3d-11da-ad31-d33d75182f1b" xmlns:xmp="http://ns.
adobe.com/xap/1.0/"><xmp:CreateDate>2018-10-16T13:05:55.767</xmp:CreateDate></rdf:Description><rdf:Desc
ription rdf:about="uuid:faf5bdd5-ba3d-11da-ad31-d33d75182f1b" xmlns:dc="http://purl.org/dc/elements/1.1
/"><dc:creator><rdf:Seq xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"><rdf:li>win10</rdf:li><
/rdf:Seq>
                        </dc:creator><dc:subject><rdf:Bag xmlns:rdf="http://www.w3.org/1999/02/22-rdf-s
yntax-ns#"><rdf:li>This is how we can do it</rdf:li></rdf:Bag>
                        </dc:subject></rdf:Description><rdf:Description rdf:about="uuid:faf5bdd5-ba3d-1
1da-ad31-d33d75182f1b" xmlns:MicrosoftPhoto="http://ns.microsoft.com/photo/1.0/"/><rdf:Description rdf:
about="uuid:faf5bdd5-ba3d-11da-ad31-d33d75182f1b" xmlns:MicrosoftPhoto="http://ns.microsoft.com/photo/1
.0/"><MicrosoftPhoto:LastKeywordXMP><rdf:Bag xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"><r
df:li>This is how we can do it</rdf:li></rdf:Bag>
```

*Figure 46: Encoded information found using the strings command.*

## 8. Evidence Captured

1. **Who is involved in the hack?**

If you refer back to figure 27 the image states that there was a data breach on the Paddy Power website and that 750.000 customers had personal details exposed.

2. **What site has been targeted?**

The Paddy Power betting website is the target for the attack.

3. **What information do the attackers know about the clients?**

If you refer back to figure 27 you will see that account balances, dates of birth, email addresses, IP addresses, names, phones numbers, physical addresses, security questions and answers, usernames and website activity.

4. **How could they attack the targeted site?**

They attackers could use the unique email address to brute force the Paddy Power database

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

One of the files was shown as deleted and replaced with a new directory that was unallocated.

One file was shown as hidden and had not sectors when using the fls command, but it was actually there.

One file was giving a different file extension to make it look like a compressed file when it was a jpg image file.

# 8. References

Sheppard, J., 2019. *Filesystems.* [Online]
Available at: https://johnshepprdnid.github.io/CompFor2019/topic03-Filesystems//talk-1/FAT-File-Systems.pdf
[Accessed 14 03 2019].

www.win.tue.nl, 2019. *www.win.tue.nl.* [Online]
Available at: https://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html
[Accessed 14 03 2019].