



ISF ASSIGNMENT 1

Ethan Roche Woodward (20094256)

I. Introduction

I was provided an image file and asked to investigate and analyse the image and to be able to answer provided questions. I did this using a range of forensic tools and forensic methods I'd learned up until now. These tools allowed me to investigate all aspects of the image and understand the ways in which files were added to and removed from the image.

II. Tools Selected

The tools I used were:

- 1) The Sleuth Kit (<https://www.sleuthkit.org/>)
- 2) Autopsy (<https://www.autopsy.com/>)
- 3) FTK Imager (<https://www.exterro.com/ftk-imager>)
- 4) FAU Toolkit
- 5) WinHex

III. Duplication

I downloaded the "Asgn1-2021.dd" image file from moodle onto my device. Before making a forensic duplicate of this image, I checked it's MD5 and SHA1 hashes using CertUtil. Along with this, before making the duplicate I used wipe.exe from the FAU toolkit to securely wipe the USB drive I'd be making the duplicate on. I used the command "wipe -w 00 \\.\X:" to fill every byte on the drive with zeros and then to ensure the USB drive was forensically secure, I did "wipe -w FF \\.\X:" to fill all the bytes on the drive with FF. Once the drive was securely wiped, I used the dd command from the FAU toolkit to make the duplicate image. The exact command that was used was "dd if=Asgn1-2021.dd of=X:\image.dd".

The MD5 and SHA1 hashes of this duplicate were then gotten and compared with the ones from the original image. These hashes matched the originals along with hashes gotten by Autopsy. A write blocker was used while getting these hashes to prevent the risk of anything being written to the drive or the image file.

```
MD5 (Asgn1-2021.dd) = e3c59ed5d55a38b37ddbacf4b1e59f71
SHA1 (Asgn1-2021.dd) = 47e76b1cd603da6b7ff35b43e6378ab87392b26f
```

(Given hashes for Asgn1-2021.dd)

```
MD5 hash of image.dd: e3c59ed5d55a38b37ddbacf4b1e59f71
SHA1 hash of image.dd: 47e76b1cd603da6b7ff35b43e6378ab87392b26f
```

(CertUtil hashes on image)

MD5	e3c59ed5d55a38b37ddbacf4b1e59f71
SHA1	47e76b1cd603da6b7ff35b43e6378ab87392b26f

(Autopsy Hashes)

IV Data Structures

To investigate the data structures on the image, I first used the fsstat command on the duplicate of the image (This will be referred to as simply the image from now on). This command shows a wide range of information about the image. With this I was able to see that the image was of file system type FAT 12. This was essential as file system types are different and may require different methods to decode.

The command listed the layout of the file system in terms of sectors. This information was then used to make the base of the file system map. The sector and cluster size was important to note too. The FAT Contents of the image were listed along with their sectors and this information was added to the map. Using “fls -r image.dd” allowed me to see the names of the files along with their inode. These inode values was used with the istat command to see and compare the sectors taken up by the files with what fsstat reported. I then added the file names to the map and also added any files that were in sectors marked as unallocated or previously deleted files.

The final map is as follows:

0	9	18	32	33	34	35	36	266	1458	1459	2299	2300	Sector 2879
1-9 FAT 0	10-18 FAT 1	19-32 Root Directory	33-33 EOF Asgn 1 Directory	34-34 EOF Folder 1	35-35 EOF The Master Plan .txt	36-36 EOF Folder 2	37-266 EOF Komatsu D61P2G-24, 2019, United Kingdom - Used crawler dozers - Mascon Ireland.jpg	267-1458 Unallocated Folder3.zip	1459- 1459 EOF Folder 4	1460-2299 EOF Map.xls.png	2300- 2300 EOF The robber ies .txt	2301-2879 Unallocated	
Reserved: 0-0 Boot Sector: 0		Data Area: 19-2879		Cluster Area: 33-2879									

Directory Entry

Here, I decoded two important parts from the file that allowed me to find and follow a cluster chain, as well as learn more information about a directory entry.

FAT table:

The File Allocation Table (FAT) is a map of the clusters in a file system. I used “blkcat -h image.dd 1” to view the hex of the first sector of FAT 0 (This is identical in sector 1 of FAT 1). I also used dd, inputting the byte size, starting sector and sector count (512, 1 and 1 respectively). This was done in case I wanted to view the hex in an external program such as WinHex, but in the end, blkcat was enough. Before starting to decode, I organised the hex into bytes, which is a group of two hexadecimal characters. This allowed for me to see what I was doing when decoding easier because decoding the FAT table requires using groups of three bytes.

Starting at the end of the hex that had contents, I took the last three bytes. The first character from the middle byte was moved to the end of the third byte and the first character from the middle byte was moved to the front of the first byte. This left me with two groups of three characters. I did this, going from end to start of the hex for every group of three bytes. This is the decoded FAT table from sector 1.

```

ff0 fff fff fff fff fff 007 008 009 00a
00b 00c 00d 00e 00f 010 011 012 013 014
015 016 017 018 019 01a 01b 01c 01d 01e
01f 020 021 022 023 024 025 026 027 028
029 02a 02b 02c 02d 02e 02f 030 031 032
033 034 035 036 037 038 039 03a 03b 03c
03d 03e 03f 040 041 042 043 044 045 046
047 048 049 04a 04b 04c 04d 04e 04f 050
051 052 053 054 055 056 057 058 059 05a
05b 05c 05d 05e 05f 060 061 062 063 064
065 066 067 068 069 06a 06b 06c 06d 06e
06f 070 071 072 073 074 075 076 077 078
079 07a 07b 07c 07d 07e 07f 080 081 082
083 084 085 086 087 088 089 08a 08b 08c
08d 08e 08f 090 091 092 093 094 095 096
097 098 099 09a 09b 09c 09d 09e 09f 0a0
0a1 0a2 0a3 0a4 0a5 0a6 0a7 0a8 0a9 0aa
0ab 0ac 0ad 0ae 0af 0b0 0b1 0b2 0b3 0b4
0b5 0b6 0b7 0b8 0b9 0ba 0bb 0bc 0bd 0be
0bf 0c0 0c1 0c2 0c3 0c4 0c5 0c6 0c7 0c8
0c9 0ca 0cb 0cc 0cd 0ce 0cf 0d1 0d2 0d3
0d4 0d5 0d6 0d7 0d8 0d9 0da 0db 0dc 0dd
0de 0df 0e0 0e1 0e2 0e3 0e4 0e5 0e6
0e7 0e8 0e9 0ea 0eb fff

```

The FF0 here simply means the FAT is reserved. From here there is four fff groups which mark both the start and end of a entry in the FAT. The 5th fff marks the first cluster of the next file, which goes from that cluster to cluster 235 (0eb) followed by another fff marking the end of the file. This is what I initially thought was the end of the FAT but as I was looking into the creation and deletion of files, I realised all of the following zeros were due to a file being deleted which resulted in all of its clusters on the FAT table being marked as 0. After this long chain of zeros from cluster 236 to cluster 1429 (which contain the deleted, and unallocated FOLDER3.zip), there was an FFF, Clusters 1429-2268, followed by another FFF. These entries referred to Folder 4(Sector 1459, MAP~1.JPG(Sectors 1460-2299) and "The robberies.txt" (Sector 2300). These clusters are the data units in the FAT filesystem and are a group of consecutive sectors (must be to a power of 2).

To follow the cluster chain, I first found what sector cluster 2 was in. I found this by simply getting the sector after the last sector of the root directory, as Cluster 2 is after the last sector of the root directory in FAT12. In the case of this image, cluster 2 was in sector 33. To get the sectors for all the clusters, I used the formula:

$(C-2) * (\text{Number of sectors per cluster}) + (\text{sector of cluster 2})$

The fsstat command showed the number of bytes per sector and cluster which in this case was the same. This meant to get the sector of cluster 7 I did $(7-2) * (1) + (33) = 38$. Knowing this let me figure out the sectors of all the FFF's before the first cluster number (37, 36, 35, 34, 33). Doing the same for 0eb gets me sector 266. I did all this as the FAT points from cluster to cluster until it reaches the end of the file, and this is a cluster chain. The cluster chain shows there is files in Sectors 33-33, 34-34, 35,35, 36-36 and 37-266. This is the same results as part of what fsstat shows.

```

33-33 (1) -> EOF
34-34 (1) -> EOF
35-35 (1) -> EOF
36-36 (1) -> EOF
37-266 (230) -> EOF

```

Directory Entry:

To decode a directory entry I first did “`blkcat -h image.dd 19 32`” to view the hex of the root directory. This is where the directory entries could all be found. I decoded the entry of TheMasterPlan.txt(THEMAS~1.txt).

To do this I extracted the 32 bytes of the file from the root directory, leaving me with:

```
54 48 45 4d 41 53 7e 31 54 58 54 00 10 02 f8 60
55 53 00 00 00 00 18 5e 55 53 04 00 4f 01 00 00
```

The first byte(54) was the first character of the file name in ASCII, which was T. Bytes 1-7 were characters 2-8 of the name which were H E M A S ~ 1. This meant the file (at least in short file name format) was named THEMAS~1. Bytes 8-10 were the extension in ASCII, which was TXT. This left me with THEMAS~1.TXT which is consistent to what `istat` showed for the file's inode.

Byte 11 contained information about the file's attributes which was 00 for this file, meaning the file was read only. Byte 12 contains reserved information for the file which was 10(16 in decimal).

The 10ths of the second of the created time was stored in byte 13 (02). The hours minutes and seconds of the file creation time was stored in the next two bytes(f8 60). This is little endian so the bytes were ordered from right to left, giving 60 f8. This was then converted from hex into binary. The binary was then split into groups of 5 bits, 6 bits and 5 bits. These groups of bits converted back into decimal gave the hours, minutes, and seconds of the creation.

```
01100 000111 11000
12      7      24
12:07:24
```

The following two bytes were also in little endian so were swapped and then converted into binary and split into groups of 7, 4 and 5 bits. The first value converted back into decimal is 41. This value is added to the Epoch (Jan 1st 1970) to get the year of creation. The next binary value converts to the decimal form of the month (For example 0011 = 3 = March) and the final value is the date. The creation time and date matched with what `istat` returned (except for the seconds).

```
0101001 1010 10101
1980+41 10    21
2021    Oct   21
```

Next was the accessed date in bytes 18 and 19, this was 00 00 which just meant that there was no access date on the file. The high 2 bytes (bytes 20-21) were also 00 00 which is always the case for FAT12. Bytes 22-23 were the time modified and are gotten in the same way that created time is gotten, and 24-25 are referring to the date modified which is also gotten through the same means as the created date. 26-27 in little endian are the bytes for the first cluster address for the file which for this file was cluster 4. This cluster's sector is sector 35. Finally, the last four bytes, also in little endian, refer to the file size in bytes.

00 00 01 4f: 335

All of the information that was gotten through this directory entry that has similar categories in the istat command were consistent with that command's results.

```
Directory Entry: 247
Allocated
File Attributes: File
Size: 335
Name: THEMAS~1.txt

Directory Entry Times:
Written:      2021-10-21 11:48:48 (GMT Summer Time)
Accessed:     0000-00-00 00:00:00 (UTC)
Created:      2021-10-21 12:07:48 (GMT Summer Time)

Sectors:
35
```

(istat screenshot)

Storage Deletion

Upon creation of Folder1, it was allocated a cluster that was then wiped to be all 0's. The size of the directory would be found only by following the cluster chain as the size field for the directory itself will always be empty.

For the creation of a file, in this case TheMasterPlan.txt , the boot sector is read from sector 0 to locate the data area, root directory and FAT structures. From here each directory entry is searched until one with the name Folder1 is found, assuming Folder1 already existed. The starting cluster of that entry is then found and read. Each entry in the directory is then searched through until an unallocated entry is found. The entry is then allocated the name TheMasterPlan.txt and it's size and creation times are also saved. The system then searches through the FAT structure to allocate clusters to the file, in this case it is allocated to cluster 4 and only cluster 4. This is because the size of the file is smaller than the cluster size. In the case of a file that is larger than the cluster size, for example KOMATS~1.JPG, the first 512 bytes are allocated to cluster 6. The FAT is then searched for another cluster, cluster 7 and allocates another 512 bytes and this continues until cluster 235 where the file is finally fully allocated. This can be seen clearly by the FAT table.

```

ff0 fff fff fff fff fff 007 008 009 00a
00b 00c 00d 00e 00f 010 011 012 013 014
015 016 017 018 019 01a 01b 01c 01d 01e
01f 020 021 022 023 024 025 026 027 028
029 02a 02b 02c 02d 02e 02f 030 031 032
033 034 035 036 037 038 039 03a 03b 03c
03d 03e 03f 040 041 042 043 044 045 046
047 048 049 04a 04b 04c 04d 04e 04f 050
051 052 053 054 055 056 057 058 059 05a
05b 05c 05d 05e 05f 060 061 062 063 064
065 066 067 068 069 06a 06b 06c 06d 06e
06f 070 071 072 073 074 075 076 077 078
079 07a 07b 07c 07d 07e 07f 080 081 082
083 084 085 086 087 088 089 08a 08b 08c
08d 08e 08f 090 091 092 093 094 095 096
097 098 099 09a 09b 09c 09d 09e 09f 0a0
0a1 0a2 0a3 0a4 0a5 0a6 0a7 0a8 0a9 0aa
0ab 0ac 0ad 0ae 0af 0b0 0b1 0b2 0b3 0b4
0b5 0b6 0b7 0b8 0b9 0ba 0bb 0bc 0bd 0be
0bf 0c0 0c1 0c2 0c3 0c4 0c5 0c6 0c7 0c8
0c9 0ca 0cb 0cc 0cd 0ce 0cf 0d1 0d2 0d3
0d4 0d5 0d6 0d7 0d8 0d9 0da 0db 0dc 0dd
0de 0df 0e0 0e1 0e2 0e3 0e3 0e4 0e5 0e6
0e7 0e8 0e9 0ea 0eb fff

```

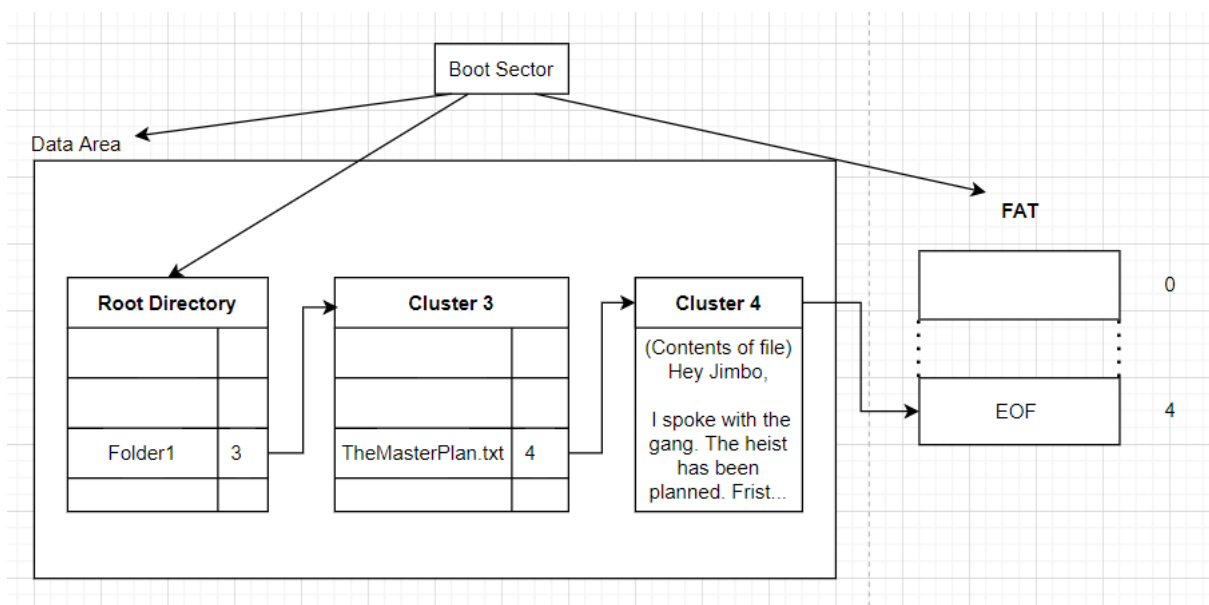
Here clusters 6-235 are allocated to the same file, which is KOMATS~1.JPG.

When it comes to deletion of a file, for example take FOLDER3.zip which was a deleted file on the image. The boot sector from sector 0 of the volume is read and the FAT structures, data area and root directories are located again. The root directory is then looked through until the directory ASGN1 is found. From here, the contents of ASGN1 are scanned until the name FOLDER3.zip is found and the starting cluster is gotten. The FAT table is used to determine the cluster chain for the file and then all of the clusters of the file (Clusters 236-1428) are all set to 0, evident by the FAT table having all 0's after cluster 235 (until cluster 1429). Along with this, the first byte in the directory for this file is changed to 0xE5 to mark the entry as unallocated. Due to this the first character of the file name is removed leaving OLDER3.zip.

```

e 5 4 f 4 c 4 4 4 5 5 2 3 3 2 0 5 a 4 9 5 0 0 0 1 0 7 0 c 7 6 1 . O L D E R 3 Z I P . . p . a
5 5 3 0 0 0 0 0 0 0 5 b 6 1 5 5 3 e c 0 0 6 c 4 f 0 9 0 0 U S . . . [ a U S . . 1 0 . .

```



(example of storage of TheMasterPlan.txt

File Recovery



When it came to recovering files, I used “fls -r image.dd” to see the files on the image, including any that might have been deleted. This command tells me the name of all the files, if they are unallocated and their inode.

```
d/d 4: Asgn1
+ d/d 230: Folder1
++ r/r 247: TheMasterplan.txt
+ r/r * 234: Folder3.zip
+ d/d 236: Folder4
++ r/r 23046: Map.xls.png
++ r/r 23049: The robberies.txt
v/v 45779: $MBR
v/v 45780: $FAT1
v/v 45781: $FAT2
V/V 45782: $OrphanFiles
```



From here I used “istat -r image.dd [inode]” on each of the results to see the starting sector and length in sectors of each file. I used the dd command with this information to extract all the files that fls showed. To extract “TheMasterPlan.txt”, I did “dd if=X:/image.dd of=./extracted/TheMasterPlan.txt bs=512 skip=35 count=1”. This took in the image file as the input and outputted to a file named TheMasterPlan.txt. The bs=512 set the byte size to 512 bytes, skip=35 just meant to skip the first 35 sectors of the image and count=1 ensured only one sector was being taken out (as the file was only taking up one sector). I tried doing similar for the ASGN1 and Folder1 directories, but the resulting files wouldn’t open and didn’t contain anything outside of the hex code. It is worth noting that TheMasterPlan.txt was in Folder1 as can be seen by the directory entry for TheMasterPlan.txt in the sector of Folder1(Sector 35). I then used FTK Imager which is a file carver to compare my extracted file to. Everything with the file was consistent with what I had extracted with dd. I also used Autopsy to ensure both FTK imager and my result was correct.

```
0 2e202020 20202020 20202010 00000000 . . . . .
16 00000000 00004962 55530300 00000000 . . . . . I b U S . . . .
32 2e2e2020 20202020 20202010 00000000 . . . . .
48 00000000 00004962 55530200 00000000 . . . . . I b U S . . . .
64 422e0074 00780074 0000000f 0013ffff B . . t . x . t . . . . .
80 fffffffffff fffffff fffffff fffffff . . . . .
96 01540068 0065004d 0061000f 00137300 . T . h . e . M . a . . . .
112 74006500 72007000 6c000000 61006e00 t . e . r . p . l . . . . a . n .
128 5448454d 41537e31 54585400 1002f860 T H E M A S ~ 1 T X T . . . .
144 55530000 0000185e 55530400 4f010000 U S . . . . ^ U S . . . . 0 . . .
```

I used icat to recover Folder3.zip as icat is used to recover deleted files. To do this I used the command “icat X:/image.dd 234 > ./extracted/Folder3.zip”. Folder3.zip contains Folder3 which contains a file called “The handoff will take place on Tuesday 16th November at 22.docx” and a png image called “Handoff.png”. This is also consistent with the findings of Autopsy and FTK Imager.

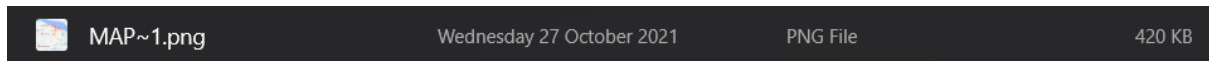
	Handoff.png	Thursday 21 October 2021	PNG File	595.01 KB
	The handoff will take place on T...	Thursday 21 October 2021	Microsoft Word Document	11.72 KB

(screenshot of extracted files in file explorer)

Name	Size	Type	Date Modif...
 Handoff.png	596	Regular File	21/10/2021...
 The handoff will t...	12	Regular File	21/10/2021...

(screenshot of FTK imager)

I tried using dd to recover Folder4, but like Folder1, this didn't result to anything worthwhile. I tried using icat to extract "Map.xls.png" but the file extracted didn't open and was also bigger than istat. Autopsy and FTK imager reported. Instead, I used "dd if=X:/image.dd of=./extracted/MAP~1.PNG" (MAP~1 was the name when doing istat on the file's inode) to recover it which worked and the image opened properly. The file recovered with dd also matched the file in Autopsy and FTK.



(File in file explorer)

Map.xls.png	1	2021-10-21 12:12:20 IST	0000-00-00 00:00:00	0000-00-00 00:00:00	2021-10-21 12:14:18 IST	429933	Allocated	Allocated	unknown
-------------	---	-------------------------	---------------------	---------------------	-------------------------	--------	-----------	-----------	---------

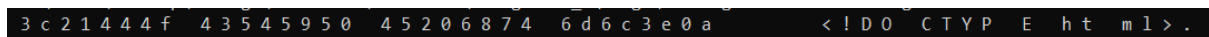
(File in Autopsy [size is in bytes])

To recover "The robberies.txt" I did the same process as recovering TheMasterPlan.txt and the resulting file was consistent with the other two tools results.

There was one directory and file that weren't listed with fls -r: Folder2 and KOMATS~1JPG. Folder2 was one of the directory names when looking at the directory entries for the ASGN1 directory as well as looking at the root directory, but there is no size allocated to this folder. I looked at the fls again and decided to test if there was something in the inode immediately after Folder1's inode and doing istat on this inode. This showed that Folder2 does exist but has a size of 0. I found KOMATS~1.JPG as it's sectors were referenced in the FAT table's cluster chain and the fsstat FAT contents section. I simply used dd to extract this file like I did with the other files. I attempted to use stegdetect on this file as it was said to be a JPG but this didn't have any results. Looking at the hex code of this file and comparing it to Gary Kessler's website of file signs (Gary C. Kessler, 2021), showed me that the file was actually a html file so I went back and extracted the file with dd again but had the output file be a .html.

```
3C 21 64 6F 63 74  <!doctype
79 70
DCI AOL HTML mail file
```

(Gary Kessler's website)



It seems like the hex was slightly modified, potentially to mask the true file type. For the last six bytes of the identifier, it seems like the first character was possibly lowered by two.

Both Folder2 and KOMATS~1.jpg were found by FTK imager but not Autopsy.

1. Who is involved in the attack?

Men named Jimbo, Muddy, Bert, somebody going by TheBoss and potentially other members of a gang are involved with this attack. This information was gotten from the "TheMasterPlan.txt" file.

2. What item has been stolen first?

The first item stolen was a Komatsu D61PXi-24 model bulldozer that potentially was stolen by Muddy and is to be passed on to Bert at the Waterford Greenway WIT West Campus Car Park.

This was gotten using information from "TheMasterPlan.txt", "KOMATS~1.jpg", and "Handoff.png".

3.What URL can the item be seen at?

The item can be seen at: <https://www.mascus.ie/construction/used-crawler-dozers/komatsu-d61px/fchkkjm5.html>

This was found in the html content of KOMATS~1.jpg.

4.What is the date for the hand-off?

The date for the hand-off is Tuesday the 16th of November at 10pm.

This was found with the file titled "The handoff will take place on Tuesday 16th November at 22.docx"

5.What has been done to mask the files on the disk?

From my investigation, it appears that Folder3.zip was deleted to mask that file and it's contents. This can be proven by the fact that it's hex code starts with 0xE5, which marks a deleted/unallocated file, along with the name of its inode being OLDER3.zip. This happens because the first byte is replaced with 0xE5, removing the first character of the file name.

KOMTS~1.jpg seems to have had the first characters of bytes 2-7 lowered by 2 to potentially mask it and prevent it's true file type from being found. The directory entry for this file also seems to have been modified to show the file was a jpg file when the file was a html file.

MAP~1.png was also masked by having a false and misleading file type attached to it which, unless view file extensions was turned on in file explorer, would have shown as simply "Map.xls".

References

Gary C. Kessler, C., 2021. *File Signatures*. [online] Garykessler.net. Available at: <https://www.garykessler.net/library/file_sigs.html> [Accessed 10 November 2021].