En el CTF de la Defcon19, en una de las partes se proponía el análisis de un pc, concretamente de la partición linux de ese pc. Hacían distintas preguntas sencillas de resolver con ftk o autopsy de un modo gráfico. Aquí haremos solo una parte (en la que había que localizar una flag que demostrase que había hackeado un pc… finalemente era una foto en el directorio root... el caso es que no se porque pero me entraron ganas de sacar la foto echándome al barro del litle endian y demás fauna de los sistemas de archivos) La finalidad no es la flag, si no tratar de comprender el sistema de archivos ext4.

Para ello yo lo haré desde un Ubuntu sencillito, previamente he convertido la imagen de encase a dd con ftk imager para trabajar más cómodo (cómodo para mí, cada uno hace lo que quiere…).

Tenemos una imagen de un disco tal que así:

```
drjekyll@drjekyll:~/Documentos/defcon$ fdisk -l adamdisk.001
Disco adamdisk.001: 50 GiB, 53687091200 bytes, 104857600 sectores
Unidades: sectores de 1 * 512 = 512 bytes
Tamaño de sector (lógico/físico): 512 bytes / 512 bytes
Tamaño de E/S (mínimo/óptimo): 512 bytes / 512 bytes
Tipo de etiqueta de disco: dos
Identificador del disco: 0xc2714295

Dispositivo    Inicio Comienzo     Final Sectores Tamaño Id Tipo
adamdisk.001p1 *          2048   1126399  1124352   549M  7 HPFS/NTFS/exFAT
adamdisk.001p2         1126400  67104767 65978368  31,5G  7 HPFS/NTFS/exFAT
adamdisk.001p3        67106816  73500671  6393856   3,1G  7 HPFS/NTFS/exFAT
adamdisk.001p4        75560958 104855551 29294594    14G  5 Extendida
adamdisk.001p5        75560960 104855551 29294592    14G 83 Linux
```

Extraigo la partición de Linux que es la que me interesa:

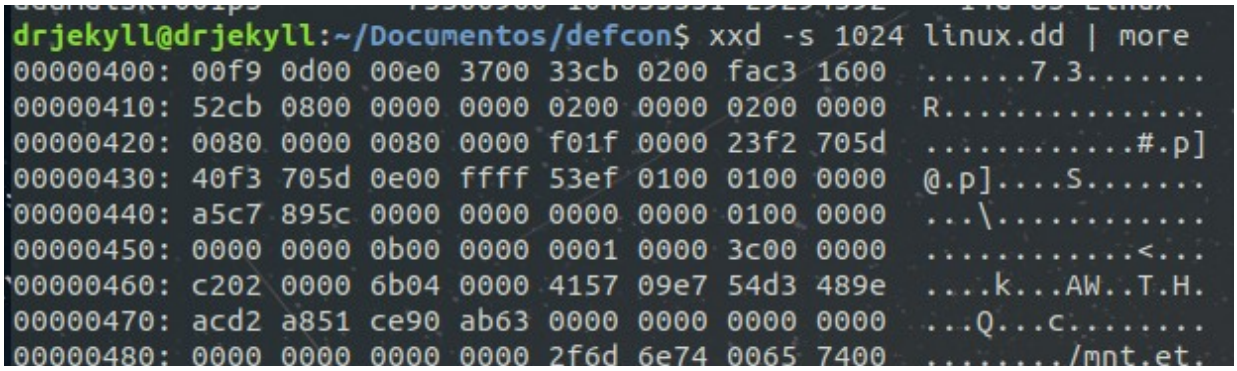
```
drjekyll@drjekyll:~/Documentos/defcon$ dd if=adamdisk.001 of=linux.dd
skip=75560960 count=104855551

29296640+0 registros leídos
29296640+0 registros escritos
14999879680 bytes (15 GB, 14 GiB) copied, 185,615 s, 80,8 MB/s
```

Supongo que se trata de un linux moderno, por lo que como sistema de archivos estaremos hablando de ext4. La principal fuente que utilizaré para hablar de ello será la wiki https://ext4.wiki.kernel.org La estructura general del ext4 es la siguiente.

Group 0 Padding	ext4 Super Block	Group Descriptors	Reserved GDT Blocks	Data Block Bitmap	inode Bitmap	inode Table	Data Blocks
1024 bytes	1 block	many blocks	many blocks	1 block	1 block	many blocks	many more block

En primer lugar necesito saber la información básica sobre los bloques y los inode; esto me lo dirá el superbloque. El superbloque se encuentra pasados los 1024 primeros bytes, entonces:



Extrayendo los bytes que me interesan los puedo interpretar con la siguiente tabla:

Offset	Size	Name	Description
0x0	__le32	s_inodes_count	Total inode count.
0x4	__le32	s_blocks_count_lo	Total block count.
0x8	__le32	s_r_blocks_count_lo	This number of blocks can only be allocated by the super-user.
0xC	__le32	s_free_blocks_count_lo	Free block count.
0x10	__le32	s_free_inodes_count	Free inode count.
0x14	__le32	s_first_data_block	First data block. This must be at least 1 for 1k-block filesystems and is typically 0 for all other block sizes.
0x18	__le32	s_log_block_size	Block size is 2 ^ (10 + s_log_block_size).
0x1C	__le32	s_log_cluster_size	Cluster size is (2 ^ s_log_cluster_size) blocks if bigalloc is enabled. Otherwise s_log_cluster_size must equal s_log_block_size.
0x20	__le32	s_blocks_per_group	Blocks per group.
0x24	__le32	s_clusters_per_group	Clusters per group, if bigalloc is enabled. Otherwise s_clusters_per_group must equal s_blocks_per_group.
0x28	__le32	s_inodes_per_group	Inodes per group.

```
drjekyll@drjekyll:~/Documentos/defcon$ xxd -s 1024 linux.dd | more

00000400: 00f9 0d00 00e0 3700 33cb 0200 fac3 1600  ......7.3.......
00000410: 52cb 0800 0000 0000 0200 0000 0200 0000  R...............
00000420: 0080 0000 0080 0000 f01f 0000 92b6 6b5d  ..............k]
00000430: 3dd8 6b5d 0d00 ffff 53ef 0100 0100 0000  =.k]....S.....…
```

En verde vemos el total de bloques, está escrito en Little Endian, luego es 0x0037e000 que nos da un total de 3661824 bloques

El tamaño de bloque lo marca como 0200000 en Little endian que siguiendo la tabla de conversión arriba mostrada es 2^(10+2)=4096 bytes

En cuanto a los bloques por grupo 00000800 equivale a 32768

Inodes por grupo 8176 (0x1ff0)

En cuanto a los inodes más populares:

inode	Purpose
0	Doesn't exist; there is no inode 0.
1	List of defective blocks.
2	Root directory.
3	User quota.
4	Group quota.
5	Boot loader.
6	Undelete directory.
7	Reserved group descriptors inode. ("resize inode")
8	Journal inode.

En el caso del inode 2 (root) estría en el grupo 0,esto lo podemos comprobar con fsstat:

```
drjekyll@drjekyll:~/Documentos/defcon$ fsstat linux.dd
[...]
Group: 0:
  Block Group Flags: [INODE_ZEROED]
  Inode Range: 1 - 8176
  Block Range: 0 - 32767
  Layout:
    Super Block: 0 - 0
    Group Descriptor Table: 1 - 2
    Group Descriptor Growth Blocks: 3 - 1026
    Data bitmap: 1027 - 1027
    Inode bitmap: 1043 - 1043
    Inode Table: 1059 - 1569
    Data Blocks: 9235 - 32767
```

Ahora queda localizar el inode en el grupo, sabemos que en ext4 los inodes pesan 256bytes, entonces en un bloque de 4096bytes habrá 16. Anteriormente habíamos extraído del superbloque que contiene 8176 inodes por block group, luego ocuparán 8176/16=512 bloques al inicio del grupo.

Sabiendo que es el inode 2 y que la tabla de inodes empieza en el 1059, como cada bloque guarda 16 inodes sabemos que nuestro inode estará en el primer bloque 1059, a partir del byte 256.

Esto es 1059*4096+256=4337920, pues leña al xxd:

```
drjekyll@drjekyll:~/Documentos/defcon$ xxd -s 4337920 linux.dd | more
00423100: ed41 0000 0010 0000 2af2 705d 3ec9 895c  .A......*.p]>..\
00423110: 3ec9 895c 0000 0000 0000 1700 0800 0000  >..\............
00423120: 0000 0800 1900 0000 0af3 0100 0400 0000  ................
00423130: 0000 0000 0000 0000 0100 0000 1324 0000  ............$..
00423140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00423150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00423160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00423170: 0000 0000 0000 0000 0000 0000 cc4b 0000  ............K..
```

La siguiente tabla muestra como leer el inode:

Offset	Size	Name	Description
0x0	__le16	i_mode	File mode. Any of:…….. 0x4000 S_IFDIR (Directory)
0x2	__le16	i_uid	Lower 16-bits of Owner UID.
0x4	__le32	i_size_lo	Lower 32-bits of size in bytes.
0x8	__le32	i_atime	Last access time, in seconds since the epoch.
0xC	__le32	i_ctime	Last inode change time, in seconds since the epoch.

0x10	__le32	i_mtime	Last data modification time, in seconds since the epoch.
0x14	__le32	i_dtime	Deletion Time, in seconds since the epoch.

Acorde a la tabla vemos que se trata de un directorio (0x41ed) y que pertenece al usuario root (UID 0000) y de tamaño 0x00000100 bytes. Podríamos saber también tanto la fecha del último acceso como de modificación (incluso si está borrado...) pero no es lo que estamos buscando ahora concretamente.

El inode de ext4 es igual que el de ext2 en sus primeros bytes, cuando se desarrolló el sistma extends no se quiso variar mucho los inodes (se duplicó el tamaño), por eso la información de extend comienza en el bloque dónde aparece el magic numer 0xF30A que nos indica que el inode usa extent tree para mapear (lo lógico en los sistemas modernos). En la tabla de abajo se ve su desglose:

Offset	Size	Name	Description
0x0	__le16	eh_magic	Magic number, 0xF30A.
0x2	__le16	eh_entries	Number of valid entries following the header.
0x4	__le16	eh_max	Maximum number of entries that could follow the header.
0x6	__le16	eh_depth	Depth of this extent node in the extent tree. 0 = this extent node points to data blocks; otherwise, this extent node points to other extent nodes. The extent tree can be at most 5 levels deep: a logical block number can be at most 2^32, and the smallest n that satisfies 4*(((blocksize - 12)/12)^n) >= 2^32 is 5.
0x8	__le32	eh_generation	Generation of the tree. (Used by Lustre, but not standard ext4).

Entonces la parte ext4 del inode nos dice lo siguiente:

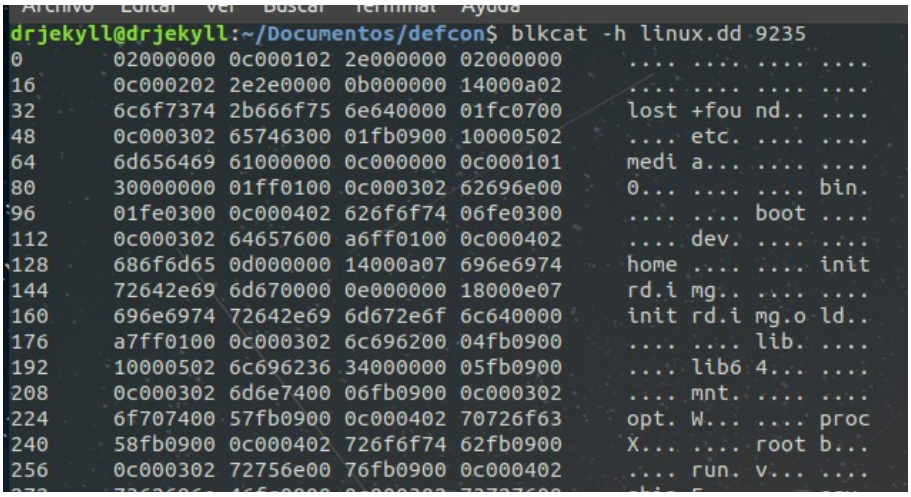
```
drjekyll@drjekyll:~/Documentos/defcon$ xxd -s 4337920 linux.dd | more

00423100: ed41 0000 0010 0000 e5cd 6b5d 3ec9 895c  .A........k]>..\
00423110: 3ec9 895c 0000 0000 0000 1700 0800 0000  >..\............
00423120: 0000 0800 1900 0000 0af3 0100 0400 0000  ................
00423130: 0000 0000 0000 0000 0100 0000 1324 0000  ............$..
00423140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00423150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Acorde a esto solo hay una entrada en el árbol (0x01)y la profundidad es 0. Veamos las “hojas” del árbol:

Offset	Size	Name	Description
0x0	__le32	ee_block	First file block number that this extent covers.
0x4	__le16	ee_len	Number of blocks covered by extent. If the value of this field is <= 32768, the extent is initialized. If the value of the field is > 32768, the extent is uninitialized and the actual extent length is ee_len - 32768. Therefore, the maximum length of a initialized extent is 32768 blocks, and the maximum length of an uninitialized extent is 32767.
0x6	__le16	ee_start_hi	Upper 16-bits of the block number to which this extent points.
0x8	__le32	ee_start_lo	Lower 32-bits of the block number to which this extent points.

En nuestro caso tenemos una logitud de 0x0001 (1) bloques y empieza en 0x00002413 (9235). Pues vamos a sacarlo:



Aquí tenemos el directorio! Desde este bloque / podemos sacar ya toda la estructura. El cómo están escritos sigue la siguiente estructura:

inode record length name length file type name and padding

Acorde a la imagen anterior del directorio que hemos obtenido:

```
240    58fb0900 0c000402 726f6f74 62fb0900       X... .... root b...
```

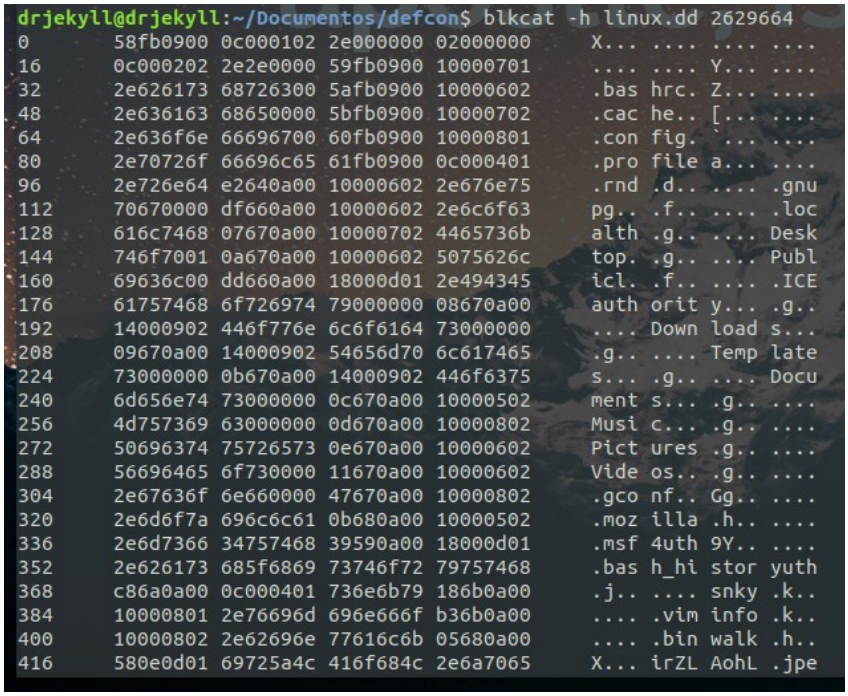
Si queremos sacar el directorio root, sabemos que está en el inode 0x0009fb58 que es el 654168. Sabiendo que hay 8176 inodes por grupo tenemos que el inode estará en el 80. fsstat lo confirma:

```
drjekyll@drjekyll:~/Documentos/defcon$ fsstat linux.dd
[...]
Group: 80:
  Block Group Flags: [INODE_ZEROED]
  Inode Range: 654081 - 662256
  Block Range: 2621440 - 2654207
  Layout:
    Data bitmap: 2621440 - 2621440
    Inode bitmap: 2621456 - 2621456
    Inode Table: 2621472 - 2621982
    Data Blocks: 2629648 - 2654207
```

654168 - 654081= 87 inodes desde el principio de la tabla, eso quiere decir que al empezar la tabla en el 2621472 y cómo cada bloque son 16 inodes estará a partir del quinto bloque, 2621477, el inicio de este bloque es el inode 654161 luego tengo que leer 7*256=1792

```
drjekyll@drjekyll:~/Documentos/defcon$ blkcat -h linux.dd 2621477
[...]
1792  ed410000 00100000 2ef2705d 40fc945c       .A.. .... ..p] @..\
1808  40fc945c 00000000 00001200 08000000       @..\ .... .... ....
1824  00000800 55000000 0af30100 04000000       .... U... .... ....
1840  00000000 00000000 01000000 20202800       .... .... .... (.
1856  00000000 00000000 00000000 00000000       .... .... .... ....
```

Como se vio antes (inode ext4… a partir del 0af3) , tenemos que el bloque es el 0x00282020 (2629664) y que tiene un tamaño de 1 bloque. Veamos el bloque:



Podemos seguir identificando directorios como hemos hecho antes Desktop, Downloads… pero entre todo esto hay una imagen, que era una de las flags del reto de la defcon 19 irZLAohL.jpeg

```
drjekyll@drjekyll:~/Documentos/defcon$ blkcat -h linux.dd 2629664
[...]
400   10000802 2e62696e 77616c6b 05680a00       .... .bin walk .h..
416   580e0d01 69725a4c 416f684c 2e6a7065       X... irZL AohL .jpe
432   672d6300 0d680a00 18000f01 2e494345       g-c. .h.. .... .ICE
```

El 01 nos indica que es un regular flie. Vemos que esta en el inode 0x000a6805 (681989). Estará a partir del grupo 83

```
drjekyll@drjekyll:~/Documentos/defcon$ fsstat linux.dd
[...]
Group: 83:
  Block Group Flags: [INODE_ZEROED]
  Inode Range: 678609 - 686784
  Block Range: 2719744 - 2752511
  Layout:
    Data bitmap: 2621443 - 2621443
    Inode bitmap: 2621459 - 2621459
    Inode Table: 2623005 - 2623515
    Data Blocks: 2719744 - 2752511
```

Dentro del grupo tenemos que 681989- 678609=3380 /16=211,25 por lo que estará en el bloque 2623005+211=2623216. Dado que ahí comineza el inode 3376 tendré que buscar 4*256=1024 más adelante

```
drjekyll@drjekyll:~/Documentos/defcon$ blkcat -h linux.dd 2623216
[...]
1024  a4810000 2bff0100 6ace6b5d 0675945c       .... +... j.k] .u.\
1040  0675945c 00000000 00000100 00010000       .u.\ .... .... ....
1056  00000800 01000000 0af30100 04000000       .... .... .... ....
1072  00000000 00000000 20000000 60aa2e00       .... ....  ... `...
1088  00000000 00000000 00000000 00000000       .... .... .... ....
```

Vemos que ocupa 0x0020 (32) bloques, desde el 0x002eaa60 (3058272). Primero comprobaré la cabecera, para ello miro en el bloque:

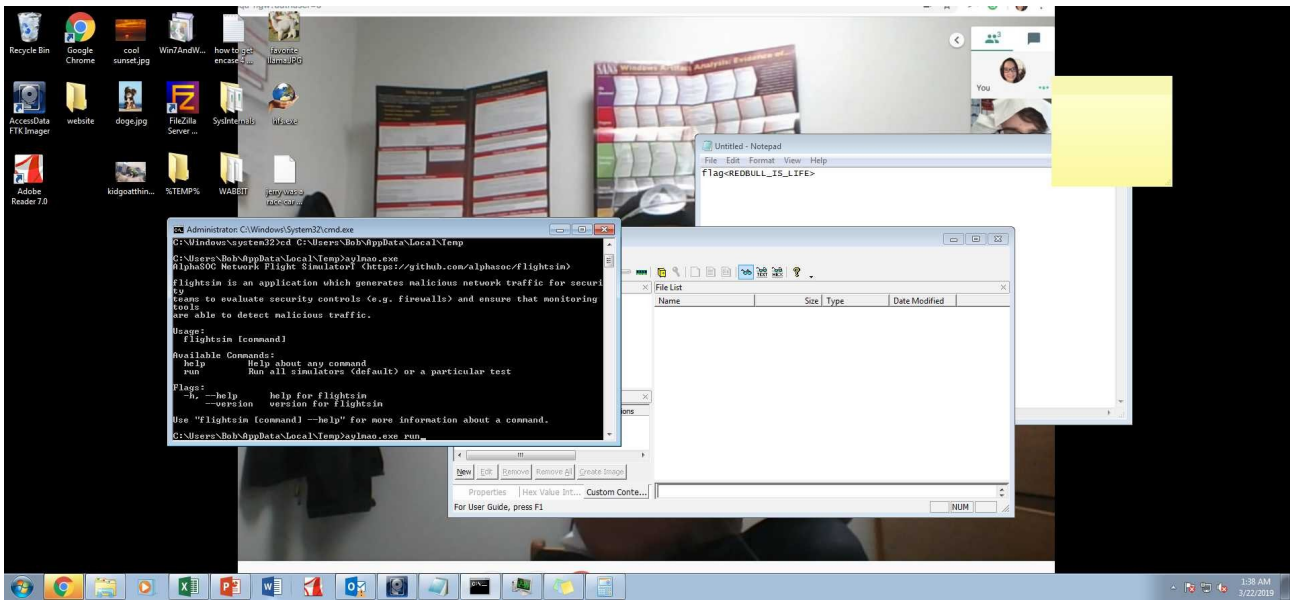
```
drjekyll@drjekyll:~/Documentos/defcon$ blkcat -h linux.dd 3058272
0       ffd8ffe0 00104a46 49460001 01000001     .... ..JF IF.. ....
16      00010000 ffdb0043 00100b0c 0e0c0a10     .... ...C .... ....
32      0e0d0e12 11101318 281a1816 16183123     .... .... (... ..1#
48      251d283a 333d3c39 33383740 485c4e40     %.(: 3=<9 387@ H\N@
64      44574537 38506d51 575f6267 68673e4d     DWE7 8PmQ W_bg hg>M
80      71797064 785c6567 63ffdb00 43011112     qypd x\eg c... C...
96      12181518 2f1a1a2f 63423842 63636363     .... /../ cB8B cccc
```

Con **ffd8ffe0** confirmamos que se trata de la imagen en fomato JPEG, que estamos buscando. Para extraerla tendría que extraer desde el 3058272 32 bloques más

```
drjekyll@drjekyll:~/Documentos/defcon$ dd if=linux.dd of=picture.jpeg bs=4096
skip=3058272 count=32

32+0 registros leídos
32+0 registros escritos
131072 bytes (131 kB, 128 KiB) copied, 0,00304803 s, 43,0 MB/s
```

Aquí la tenemos! En teoría el malo desde un Kali había “juankeado” un Windows 7 y se había guardado esta foto:



Todo esto también lo podíamos haber sacado montando la partición pero no habríamos tenido la oportunidad de destripar un ext4 a nivel de leer Little Endian como un pro jejeje

```
drjekyll@drjekyll:~/Documentos/defcon$ sudo mount -o loop linux.dd /mnt/
```

Pero no era la idea; esa otra forma está disponible aquí junto con el resto del ctf
https://github.com/drj3ky11/DefCON19ctf

Espero que haya servido para explicar un poco el sistema de archivos ext4 y como bucear a través de él mediante una de las formas más básicas posibles.