JUST CTF 2020

ROXY Write-up

- The description says “Help me! Find little toy roxy.”, the attached files was “roxy.jpg”, download the file and let’s start working.
- First, we use “***file***” command to determine the file type as following:

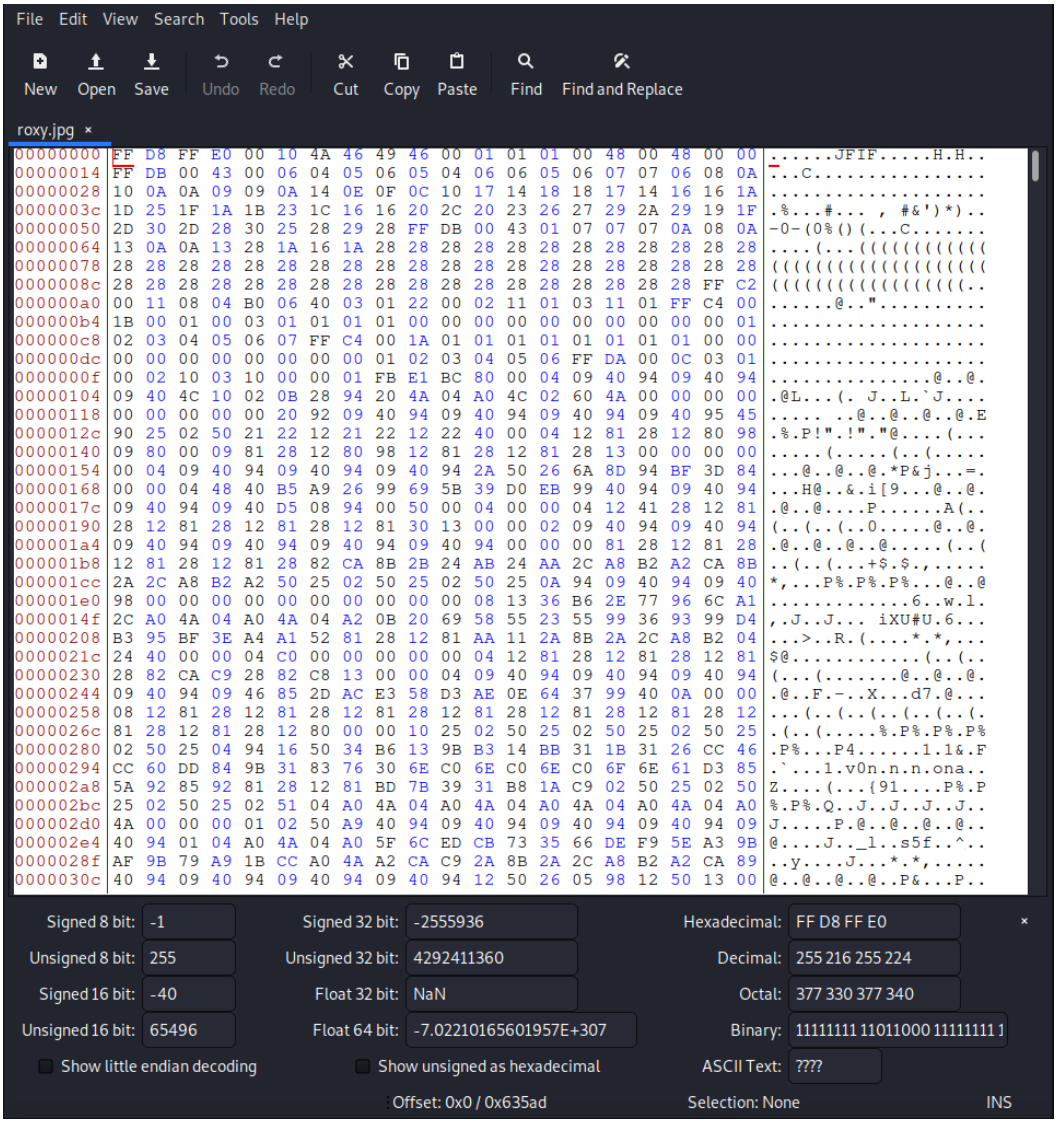
```
FILE(1)                          BSD General Commands Manual                          FILE(1)

NAME
     file – determine file type

SYNOPSIS
     file [-bcdEhiklLNnprsSvzZ0] [--apple] [--extension] [--mime-encoding] [--mime-type] [-e testname]
          [-F separator] [-f namefile] [-m magicfiles] [-P name=value] file ...
     file -C [-m magicfiles]
     file [--help]
```

```
└─$ $file roxy.jpg
roxy.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72×72, segment length 16, progressive, pr
ecision 8, 1600×1200, components 3
┌──[cryp70n@KALI]─[~/Downloads/justCTF]
└─$ $
```

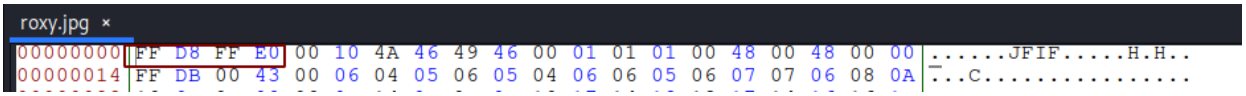
The output indicates that “roxy.jpg” is a *JPEG image* file, let’s dump the file contents into hexadecimal format. For this to be done, you can use one of the hexadecimal editor like :“hexdump” , “xxd”, or the one I prefer “bless”,which has a graphical interface.

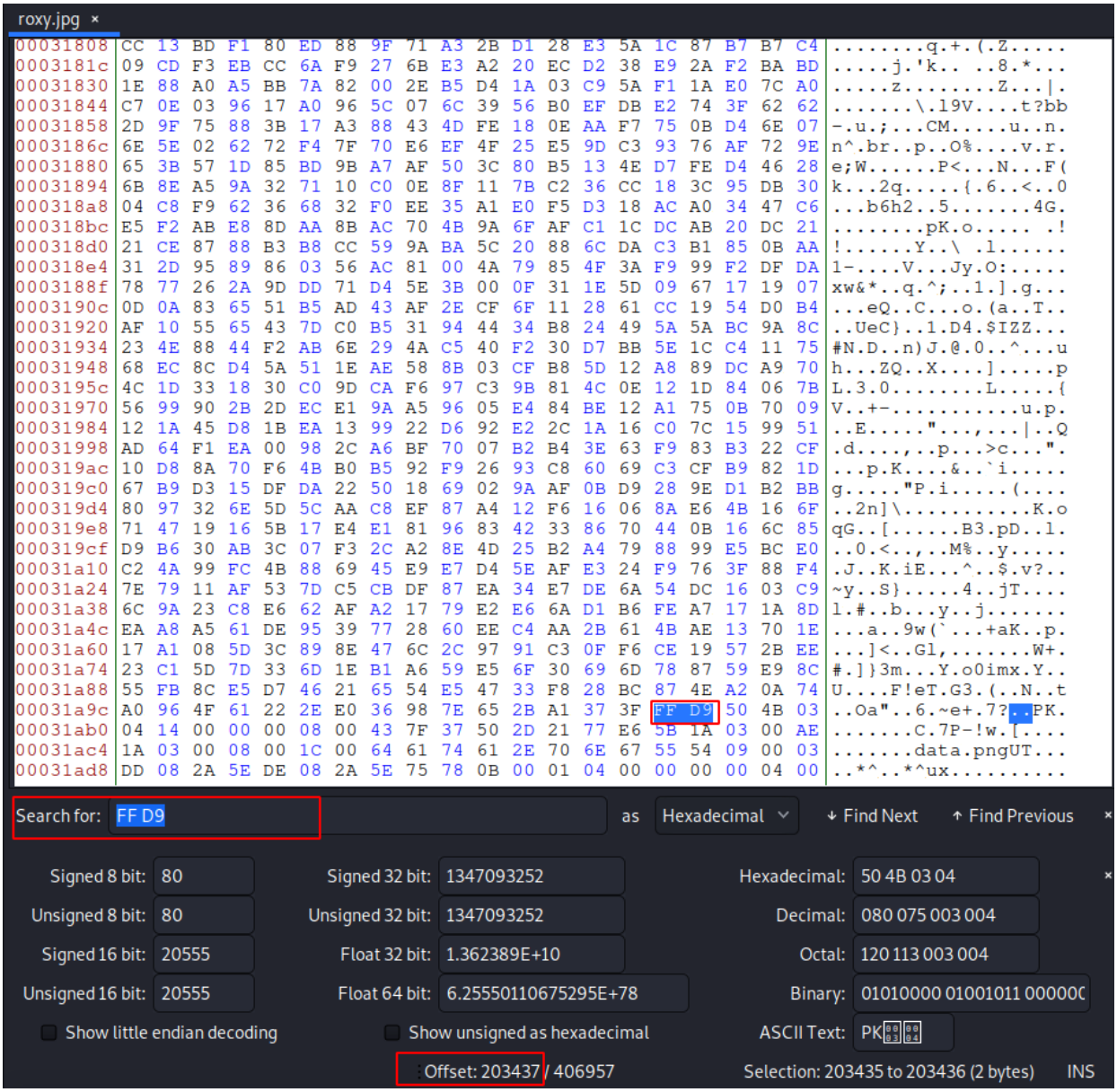


- Thing you have to know is the Magic numers, which are the first bits of a file that uniquely identify the type of the file.
- The following table contains a samples of these magic number :

File type	Typical extension	Hex digits xx = variable	Ascii digits . = not an ascii char
Bitmap format	.bmp	42 4d	BM
FITS format	.fits	53 49 4d 50 4c 45	SIMPLE
GIF format	.gif	47 49 46 38	GIF8
Graphics Kernel System	.gks	47 4b 53 4d	GKSM
IRIS rgb format	.rgb	01 da	..
ITC (CMU WM) format	.itc	f1 00 40 bb	....
JPEG File Interchange Format	.jpg	ff d8 ff e0	....
NIFF (Navy TIFF)	.nif	49 49 4e 31	IIN1
PM format	.pm	56 49 45 57	VIEW
PNG format	.png	89 50 4e 47	.PNG
Postscript format	.[e]ps	25 21	%!
Sun Rasterfile	.ras	59 a6 6a 95	Y.j.
Targa format	.tga	xx xx xx	...
TIFF format (Motorola - big endian)	.tif	4d 4d 00 2a	MM.*
TIFF format (Intel - little endian)	.tif	49 49 2a 00	II*.
X11 Bitmap format	.xbm	xx xx	
XCF Gimp file structure	.xcf	67 69 6d 70 20 78 63 66 20 76	gimp xcf
Xfig format	.fig	23 46 49 47	#FIG

As the table shown, JPEG images start with “FF D8 FF E0”, and this can be obviously seen in ***bless.***





- To determine the end of the JPEG image, it also has EOI “End Of Image” bytes, which are “FF D9”.
- Using *bless* , search for “FF D9” , and get the offset of that pattern, which is “203437”.

- Now we know that the first “203437” bytes of “roxy.jpg” are the actual image, and the rest of the file is extra data. In order to extract the actual image from the full file, we use the following command.

```
┌──[cryp70n@KALI]─[~/Downloads/justCTF]
└─$head -c +203437 roxy.jpg | tee roxyNoData.jpg
```

```
┌──[cryp70n@KALI]─[~/Downloads/justCTF]
└─$ls -l roxyNoDat.jpg
-rw-r--r-- 1 cryp70n cryp70n 203437 Apr  5 03:05 roxyNoDat.jpg
```

- Now, we can search for embedded files that the image may hold using *binwalk* tool.

```
┌──[cryp70n@KALI]─[~/Downloads/justCTF]
└─$binwalk roxy.jpg

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
203437        0x31AAD         Zip archive data, at least v2.0 to extract, compressed size: 203355, uncompressed si
ze: 203438, name: data.png
406936        0x63598         End of Zip archive, footer length: 22
```

The image holds ZIP archive, which contains “data.png” file, to extract the embedded data, we add option *-e* binwalk.

- *The hint for this task was “Reverse my name”, Roxy becomes y**XOR**, so let’s try to perform XORing between “roxyNoDat.jpg” and “data.png”, to do so , I’ve used the following python script which reads images as bytes and perform XOR operation byte wise.*

```
def str_xor(data, key):
        for i in range(len(data)):
         data[i] ^= key[i % len(key)]
        return data


key  = bytearray(open('data.png', 'rb').read())
data = bytearray(open('roxyNoDat.jpg',  'rb').read())
encoded = str_xor(data, key)
open("t.jpg", "wb").write(encoded)
```

That was the result of XORing the two files, I took this image to StegSolver , and I’ve got the flag.