

## LinkedIn

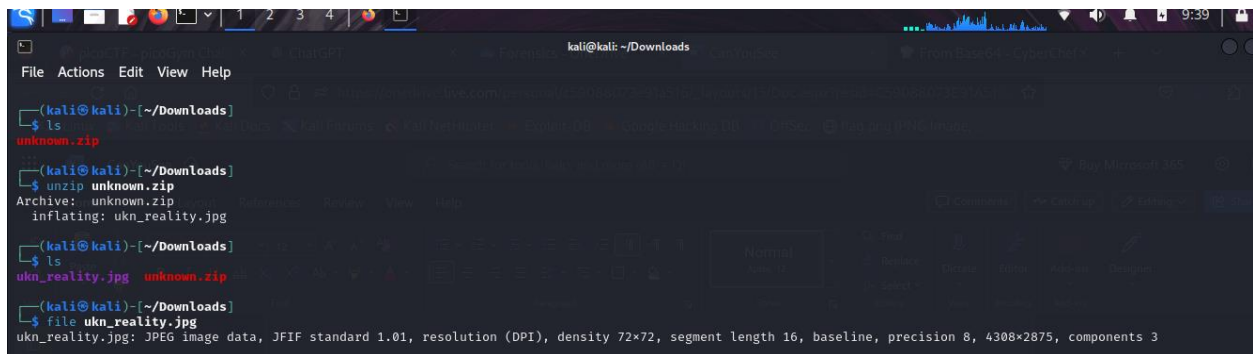
# Description

How about some hide and seek? Download this file here.

***Hint:** How can you view the information about the picture? If something isn't in the expected form, maybe it deserves attention?*

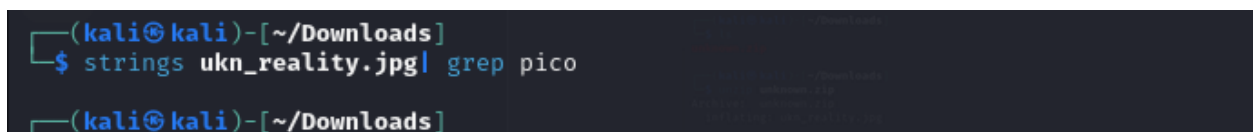
## Solution

I started by downloading the provided ZIP file and extracted its contents. Inside the archive, I found an image file named `ukn_reality.jpg`. After inspecting the file with the **file** command, it confirmed that it was a standard JPEG image.

A terminal window on a Kali Linux system. The user is in the ~/Downloads directory. They run 'ls' and see 'unknown.zip'. Then they run 'unzip unknown.zip', which extracts 'ukn\_reality.jpg'. They run 'ls' again to confirm the file is there. Finally, they run 'file ukn\_reality.jpg', which outputs: 'ukn\_reality.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72x72, segment length 16, baseline, precision 8, 4308x2875, components 3'.

```
(kali@kali)-[~/Downloads]
$ ls
unknown.zip
(kali@kali)-[~/Downloads]
$ unzip unknown.zip
Archive: unknown.zip
  inflating: ukn_reality.jpg
(kali@kali)-[~/Downloads]
$ ls
ukn_reality.jpg  unknown.zip
(kali@kali)-[~/Downloads]
$ file ukn_reality.jpg
ukn_reality.jpg: JPEG image data, JFIF standard 1.01, resolution (DPI), density 72x72, segment length 16, baseline, precision 8, 4308x2875, components 3
```

Curious about any hidden information, I ran **strings** on the image, but I didn't find anything immediately useful.

A terminal window showing the command 'strings ukn\_reality.jpg | grep pico' being entered. The output is not visible.

```
(kali@kali)-[~/Downloads]
$ strings ukn_reality.jpg | grep pico
(kali@kali)-[~/Downloads]
```

I then used **binwalk** to look for embedded data or anomalies, but it only showed the JPEG header with no additional embedded files.

```
(kali@kali)-[~/Downloads]
$ binwalk -e ukn_reality.jpg

DECIMAL      HEXADECMAL    DESCRIPTION
-----
0            0x0          JPEG image data, JFIF standard 1.01

(kali@kali)-[~/Downloads]
$ ls
ukn_reality.jpg  unknown.zip
```

Next, I used **exiftool** to examine the image's metadata. The metadata contained an interesting Attribution URL field, which looked suspiciously like base64-encoded data.

```
(kali@kali)-[~/Downloads]
$ exiftool ukn_reality.jpg
ExifTool Version Number      : 12.76
File Name                    : ukn_reality.jpg
Directory                   : .
File Size                    : 2.3 MB
File Modification Date/Time  : 2024:03:12 00:05:55+00:00
File Access Date/Time       : 2024:03:12 00:05:55+00:00
File Inode Change Date/Time  : 2025:01:07 09:34:21+00:00
File Permissions             : -rw-r--r--
File Type                   : JPEG
File Type Extension         : jpg
MIME Type                   : image/jpeg
JFIF Version                : 1.01
Resolution Unit             : inches
X Resolution                 : 72
Y Resolution                 : 72
XMP Toolkit                 : Image::ExifTool 11.88
Attribution URL             : cGljb0NURntNRTc0RDQ3QV9ISUREM05fNmE5ZjVhYzR9Cg==
Image Width                  : 4308
Image Height                 : 2875
Encoding Process             : Baseline DCT, Huffman coding
Bits Per Sample              : 8
Color Components             : 3
Y Cb Cr Sub Sampling         : YCbCr4:2:0 (2 2)
Image Size                   : 4308x2875
Megapixels                   : 12.4

(kali@kali)-[~/Downloads]
$
```

The decoded string turned out to be the flag: **picoCTF{ME74D47A\_HIDD3N\_6a9f5ac4}**. This hidden information, embedded within the image's metadata, was the key to solving the challenge.

```
(kali@kali)-[~/Downloads]
$ echo "cGljb0NURntNRTc0RDQ3QV9ISUREM05fNmE5ZjVhYzR9Cg==" | base64 -d
picoCTF{ME74D47A_HIDD3N_6a9f5ac4}

(kali@kali)-[~/Downloads]
$
```