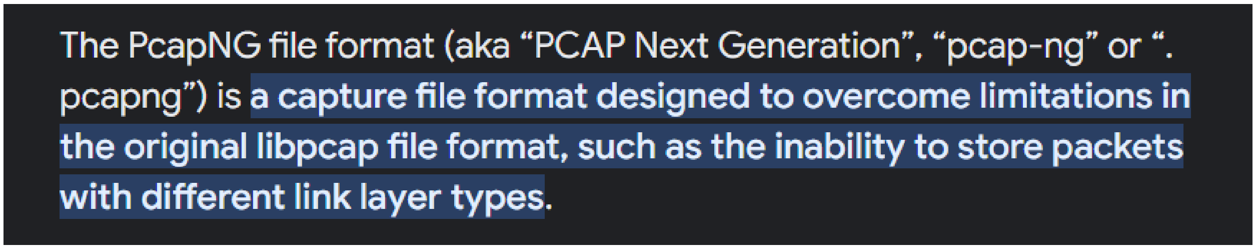
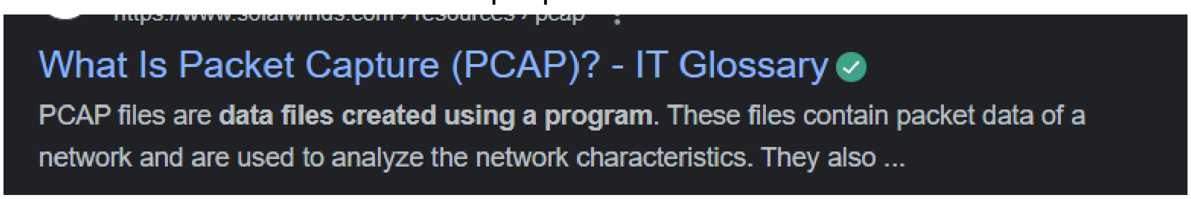
Trivial FTP

This one didn’t have much info on it, simply had a file *tftp.pcapng*
I immediately looked up what the file extension .pcapng meant



The PcapNG file format (aka “PCAP Next Generation”, “pcap-ng” or “. pcapng”) is a capture file format designed to overcome limitations in the original libpcap file format, such as the inability to store packets with different link layer types.

I decided to search the full form of pcap and saw this :



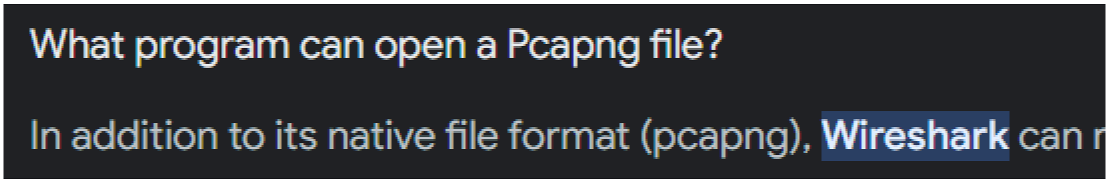
https://www.solarwinds.com › resources › pcap

What Is Packet Capture (PCAP)? - IT Glossary

PCAP files are **data files created using a program**. These files contain packet data of a network and are used to analyze the network characteristics. They also ...

Not sure, but I think it's a file containing network packets.

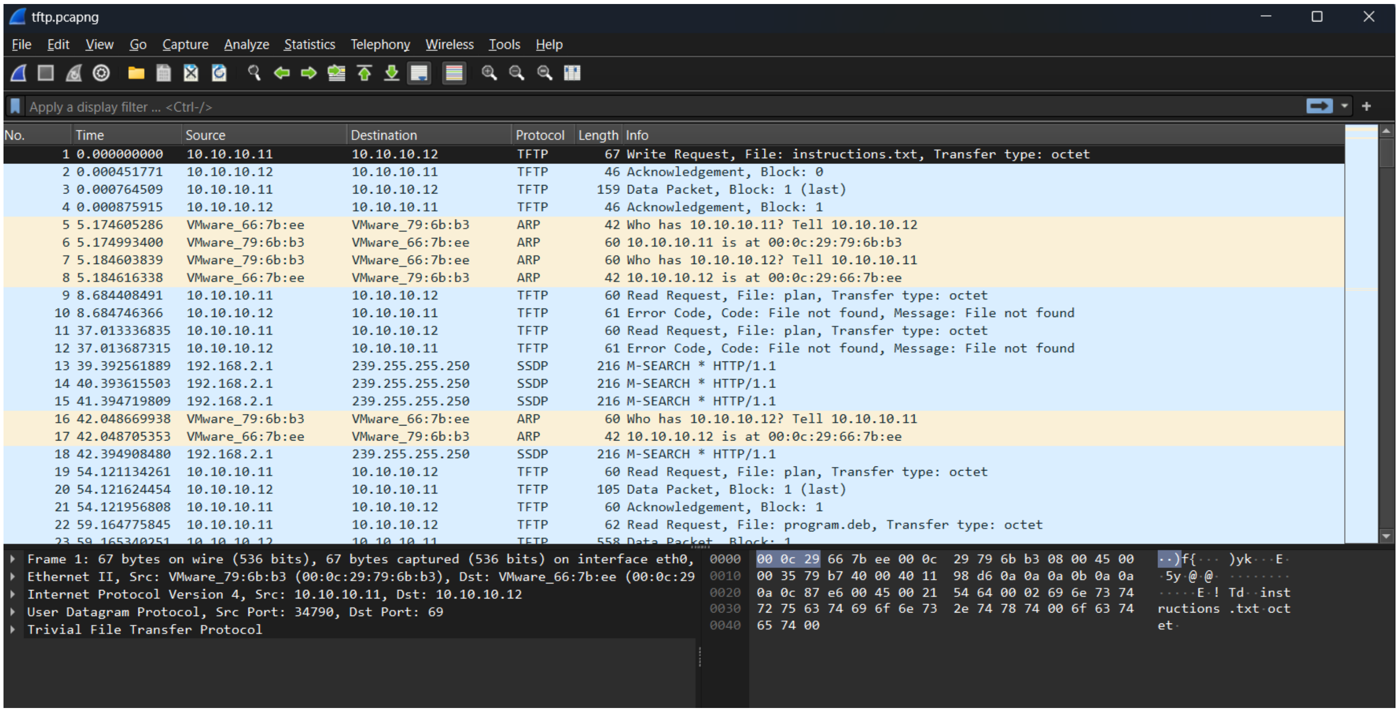
I searched up how to open these



What program can open a Pcapng file?

In addition to its native file format (pcapng), Wireshark can r

And soon installed and ran wireshark.



tftp.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

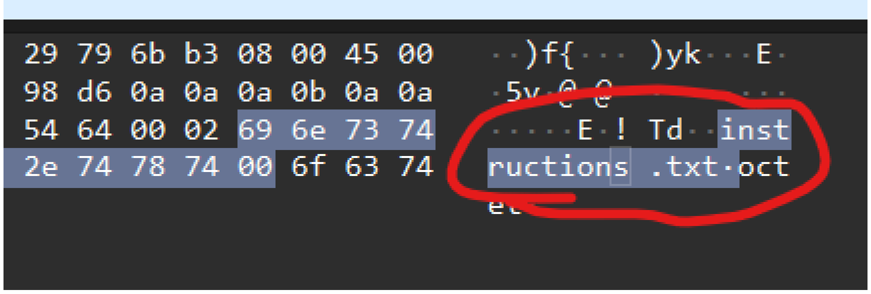
```
No.  Time          Source             Destination        Protocol Length Info
   1 0.000000000   10.10.10.11        10.10.10.12        TFTP       67 Write Request, File: instructions.txt, Transfer type: octet
   2 0.000451771   10.10.10.12        10.10.10.11        TFTP       46 Acknowledgement, Block: 0
   3 0.000764509   10.10.10.11        10.10.10.12        TFTP      159 Data Packet, Block: 1 (last)
   4 0.000875915   10.10.10.12        10.10.10.11        TFTP       46 Acknowledgement, Block: 1
   5 5.174605286   VMware_66:7b:ee    VMware_79:6b:b3    ARP        42 Who has 10.10.10.11? Tell 10.10.10.12
   6 5.174993400   VMware_79:6b:b3    VMware_66:7b:ee    ARP        60 10.10.10.11 is at 00:0c:29:79:6b:b3
   7 5.184603839   VMware_79:6b:b3    VMware_66:7b:ee    ARP        60 Who has 10.10.10.12? Tell 10.10.10.11
   8 5.184616338   VMware_66:7b:ee    VMware_79:6b:b3    ARP        42 10.10.10.12 is at 00:0c:29:66:7b:ee
   9 8.684408491   10.10.10.11        10.10.10.12        TFTP       60 Read Request, File: plan, Transfer type: octet
  10 8.684746366   10.10.10.12        10.10.10.11        TFTP       61 Error Code, Code: File not found, Message: File not found
  11 37.013336835  10.10.10.11        10.10.10.12        TFTP       60 Read Request, File: plan, Transfer type: octet
  12 37.013687315  10.10.10.12        10.10.10.11        TFTP       61 Error Code, Code: File not found, Message: File not found
  13 39.392561889  192.168.2.1        239.255.255.250    SSDP      216 M-SEARCH * HTTP/1.1
  14 40.393615503  192.168.2.1        239.255.255.250    SSDP      216 M-SEARCH * HTTP/1.1
  15 41.394719809  192.168.2.1        239.255.255.250    SSDP      216 M-SEARCH * HTTP/1.1
  16 42.048669938  VMware_79:6b:b3    VMware_66:7b:ee    ARP        60 Who has 10.10.10.12? Tell 10.10.10.11
  17 42.048705353  VMware_66:7b:ee    VMware_79:6b:b3    ARP        42 10.10.10.12 is at 00:0c:29:66:7b:ee
  18 42.394908480  192.168.2.1        239.255.255.250    SSDP      216 M-SEARCH * HTTP/1.1
  19 54.121134261  10.10.10.11        10.10.10.12        TFTP       60 Read Request, File: plan, Transfer type: octet
  20 54.121624454  10.10.10.12        10.10.10.11        TFTP      105 Data Packet, Block: 1 (last)
  21 54.121956808  10.10.10.11        10.10.10.12        TFTP       60 Acknowledgement, Block: 1
  22 59.164775845  10.10.10.11        10.10.10.12        TFTP       62 Read Request, File: program.deb, Transfer type: octet
  23 59.165340251  10.10.10.12        10.10.10.11        TFTP      558 Data Packet, Block: 1
```

```
> Frame 1: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface eth0,
> Ethernet II, Src: VMware_79:6b:b3 (00:0c:29:79:6b:b3), Dst: VMware_66:7b:ee (00:0c:29
> Internet Protocol Version 4, Src: 10.10.10.11, Dst: 10.10.10.12
> User Datagram Protocol, Src Port: 34790, Dst Port: 69
> Trivial File Transfer Protocol
```

```
0000  00 0c 29 66 7b ee 00 0c  29 79 6b b3 08 00 45 00   ··)f{··· )yk···E·
0010  00 35 79 b7 40 00 40 11  98 d6 0a 0a 0a 0b 0a 0a   ·5y·@·@· ········
0020  0a 0c 87 e6 00 45 00 21  54 64 00 02 69 6e 73 74   ·····E·! Td··inst
0030  72 75 63 74 69 6f 6e 73  2e 74 78 74 00 6f 63 74   ructions .txt·oct
0040  65 74 00                                           et·
```

I had no clue what was going on

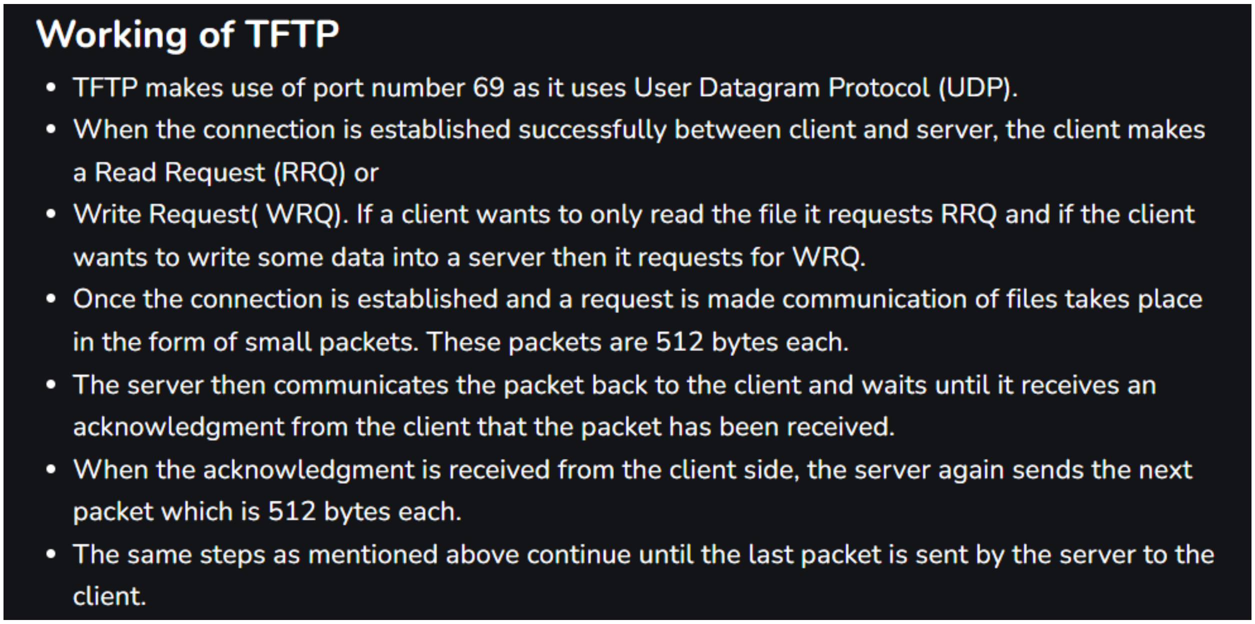
I noticed certain file names here and there :



```
29 79 6b b3 08 00 45 00   ··)f{···  )yk···E·
98 d6 0a 0a 0a 0b 0a 0a   ·5y·@·@·  ········
54 64 00 02 69 6e 73 74   ·····E·! Td··inst
2e 74 78 74 00 6f 63 74   ructions .txt·oct
                          et·
```

Maybe I was to extract these files from this thing somehow ?

After a few more minutes of going through it,
I noticed a lot of them had “tftp” as their protocol. Coincidentally the challenge was named the same. I decided to look up exactly what tftp is and how it works.

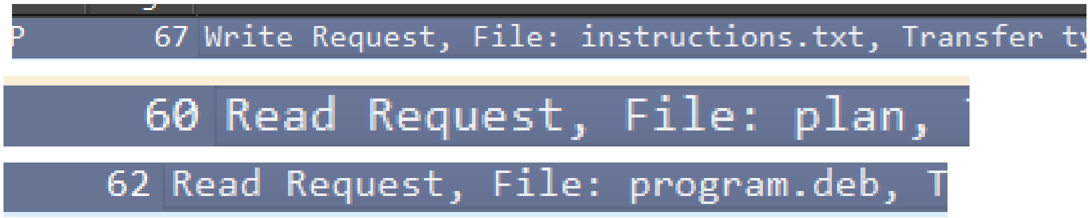


Working of TFTP

- TFTP makes use of port number 69 as it uses User Datagram Protocol (UDP).
- When the connection is established successfully between client and server, the client makes a Read Request (RRQ) or
- Write Request(WRQ). If a client wants to only read the file it requests RRQ and if the client wants to write some data into a server then it requests for WRQ.
- Once the connection is established and a request is made communication of files takes place in the form of small packets. These packets are 512 bytes each.
- The server then communicates the packet back to the client and waits until it receives an acknowledgment from the client that the packet has been received.
- When the acknowledgment is received from the client side, the server again sends the next packet which is 512 bytes each.
- The same steps as mentioned above continue until the last packet is sent by the server to the client.

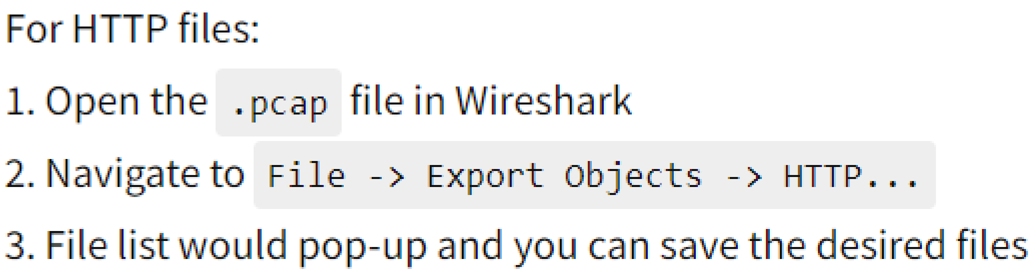
What I gathered was that it was a simple file transfer protocol that transfers packets of 512 byte size back and forth via read requests, write requests and acknowledgements.

So going through the thing, I noticed that, at the read requests and write requests certain files were being exchanged ?



```
P        67 Write Request, File: instructions.txt, Transfer ty
       60 Read Request, File: plan, T
     62 Read Request, File: program.deb, T
```

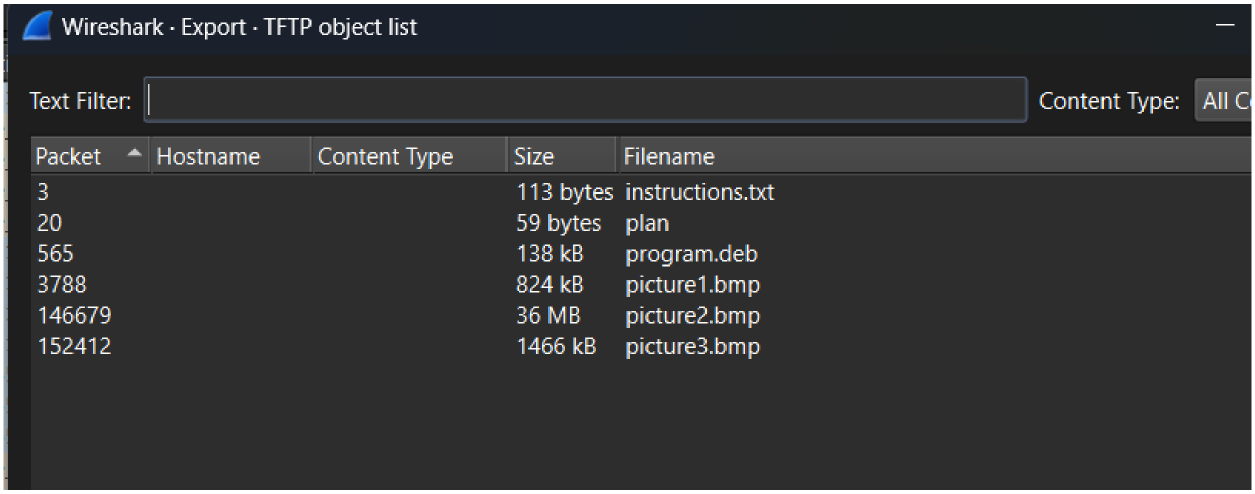
I searched up how to extract files from a pcapng in wireshark and reached this site
Where I saw this :



For HTTP files:

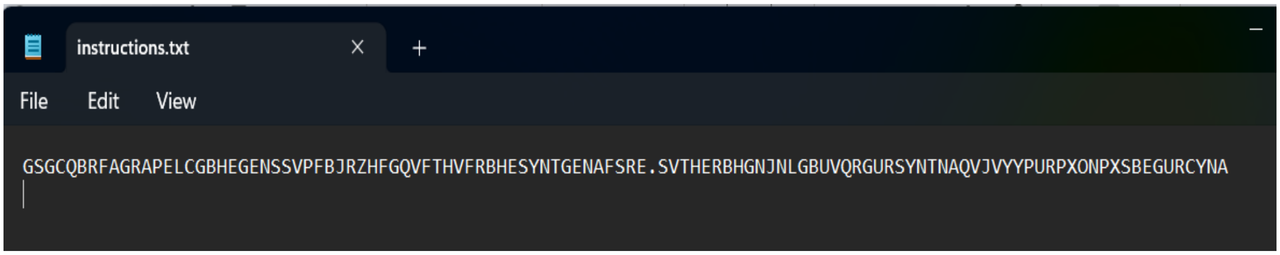
1. Open the `.pcap` file in Wireshark
2. Navigate to `File -> Export Objects -> HTTP...`
3. File list would pop-up and you can save the desired files

I did the same, but after export objects, hit tftp instead, and :

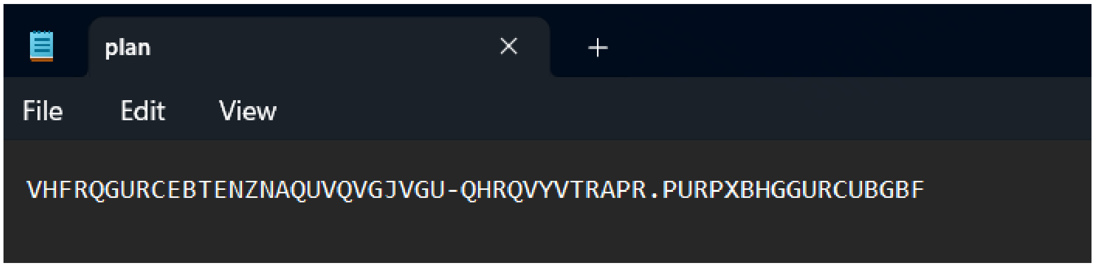


Packet	Hostname	Content Type	Size	Filename
3			113 bytes	instructions.txt
20			59 bytes	plan
565			138 kB	program.deb
3788			824 kB	picture1.bmp
146679			36 MB	picture2.bmp
152412			1466 kB	picture3.bmp

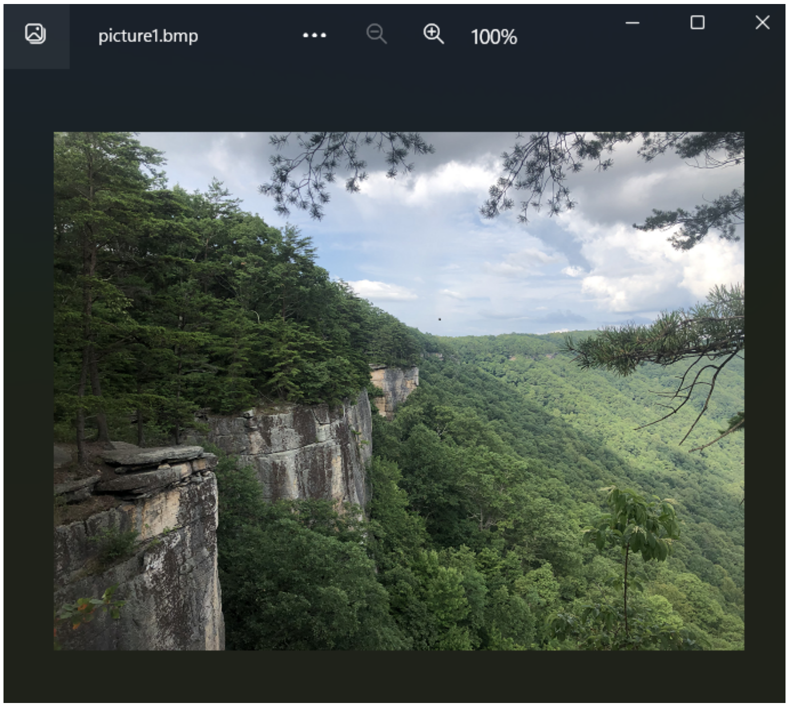
I saved them and checked them out.

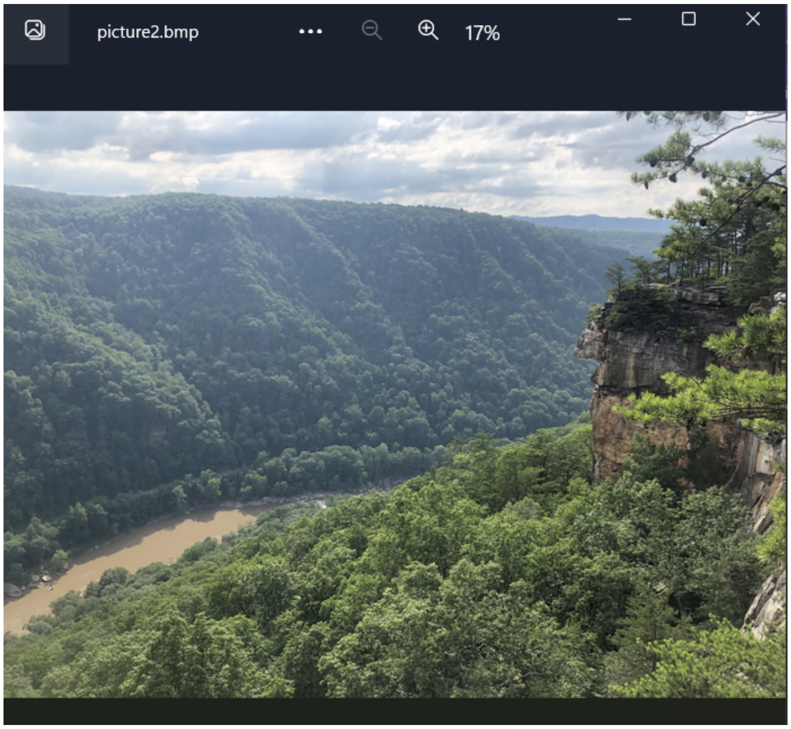


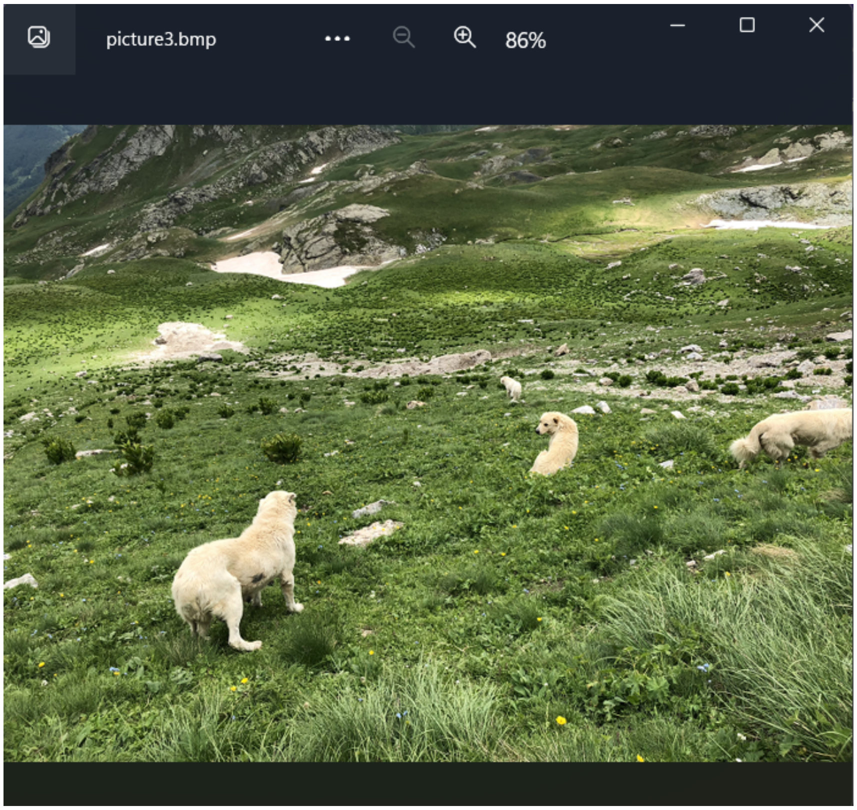
```
GSGCQBRFAGRAPELCGBHEGENSSVPFBJRZHFGQVFTHVFRBHESYNTGENAFSRE.SVTHERBHGNJNLGBUVQRGURSYNTNAQVJVYYPURPXONPXSBEGURCYNA
```



```
VHFRQGURCEBTENZNAQUVQVGJVGU-QHRQVYVTRAPR.PURPXBHGGURCUBGBF
```


picture1.bmp
100%


picture2.bmp
17%



When I rot13’d the text in instructions.txt and plan, I got these :
TFTPDOESNTENCRYPTOURTRAFFICSOWEMUSTDISGUISEOURFLAGTRANSFER
.FIGUREOUTAWAYTOHIDETHEFLAGANDIWILLCHECKBACKFORTHEPLAN

IUSEDTHEPROGRAMANDHIDITWITH-DUEDILIGENCE.CHECKOUTTHEPHOTOS

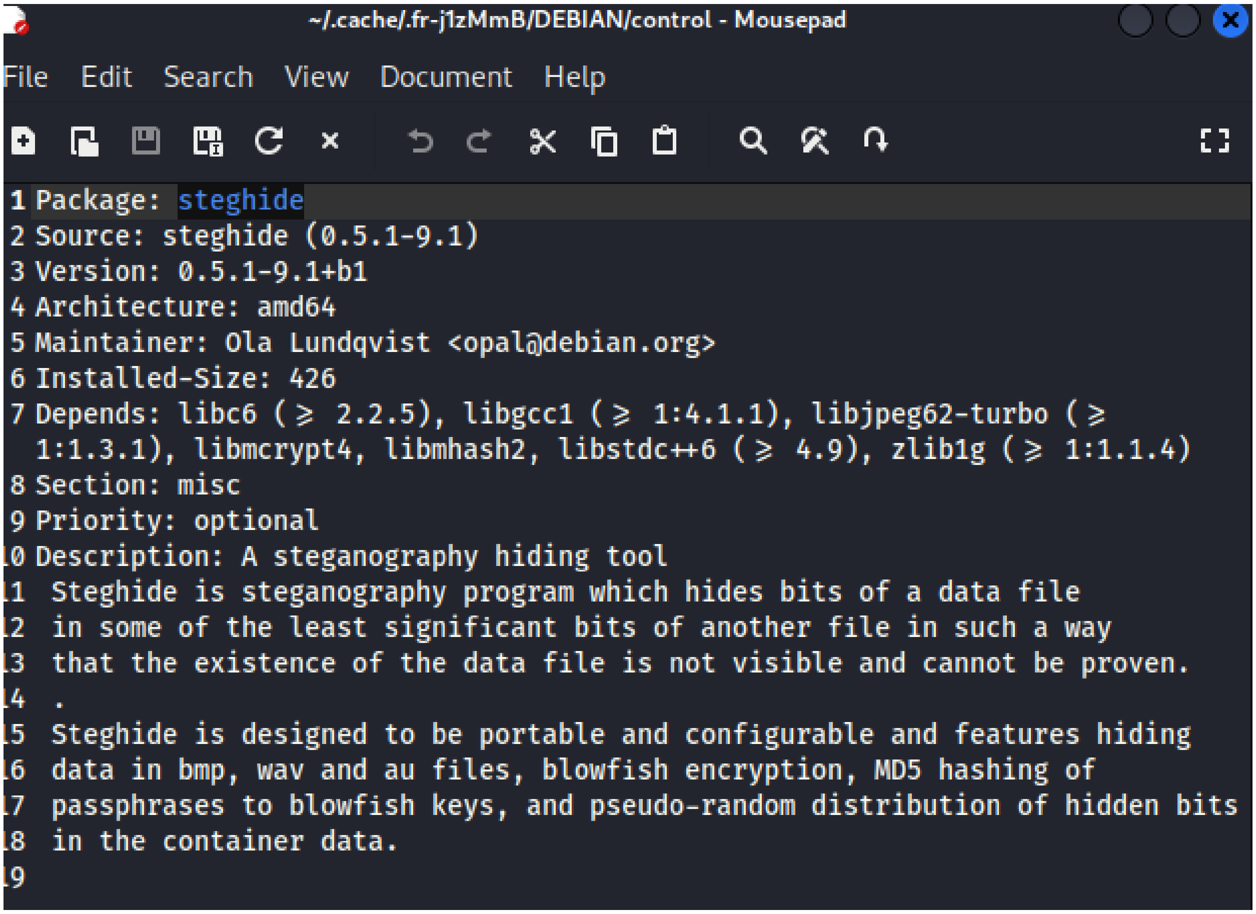
i.e

TFTP DOESNT ENCRYPT OUR TRAFFIC SO WE MUST DISGUISE OUR FLAG TRANSFER. FIGURE OUT A WAY TO HIDE THE FLAG AND I WILL CHECK BACK FOR THE PLAN.

I USED THE PROGRAM AND HID IT WITH - DUEDILIGENCE. CHECKOUT THE PHOTOS

Now, from the files that were downloaded, there was also one **program.deb** and three pictures.

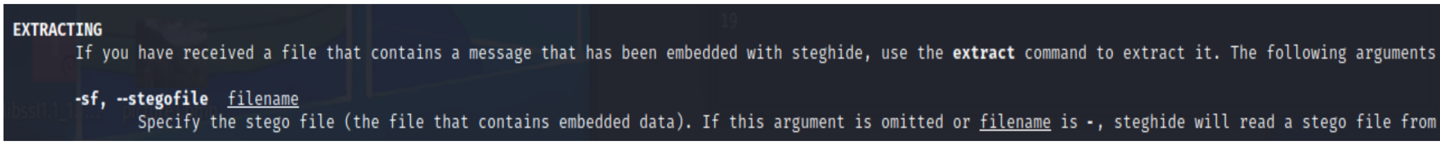
I proceeded to install that deb file and when I checked its contents I saw this :


~/.cache/.fr-j1zMmB/DEBIAN/control - Mousepad
File Edit Search View Document Help

```
Package: steghide
Source: steghide (0.5.1-9.1)
Version: 0.5.1-9.1+b1
Architecture: amd64
Maintainer: Ola Lundqvist <opal@debian.org>
Installed-Size: 426
Depends: libc6 (≥ 2.2.5), libgcc1 (≥ 1:4.1.1), libjpeg62-turbo (≥
1:1.3.1), libmcrypt4, libmhash2, libstdc++6 (≥ 4.9), zlib1g (≥ 1:1.1.4)
Section: misc
Priority: optional
Description: A steganography hiding tool
 Steghide is steganography program which hides bits of a data file
 in some of the least significant bits of another file in such a way
 that the existence of the data file is not visible and cannot be proven.
 .
 Steghide is designed to be portable and configurable and features hiding
 data in bmp, wav and au files, blowfish encryption, MD5 hashing of
 passphrases to blowfish keys, and pseudo-random distribution of hidden bits
 in the container data.
```

Which led me to believe that this program was simply only installing steghide,
And I was to use steghide to extract data from the pictures.

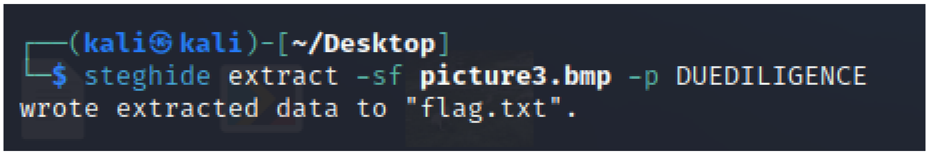
Man steghide gave me this



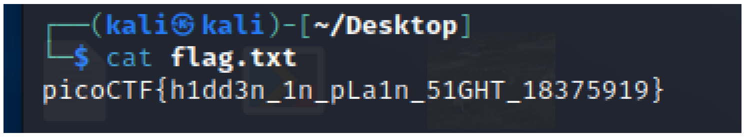
```
EXTRACTING
       If you have received a file that contains a message that has been embedded with steghide, use the extract command to extract it. The following arguments

       -sf, --stegofile filename
              Specify the stego file (the file that contains embedded data). If this argument is omitted or filename is -, steghide will read a stego file from
```

I proceeded to use steghide on the pictures one by one with the passphrase DUEDILLIGENCE, as it was mentioned in the plan file.. and boom :



```
┌──(kali㉿kali)-[~/Desktop]
└─$ steghide extract -sf picture3.bmp -p DUEDILIGENCE
wrote extracted data to "flag.txt".
```



```
┌──(kali㉿kali)-[~/Desktop]
└─$ cat flag.txt
picoCTF{h1dd3n_1n_pLa1n_51GHT_18375919}
```