



Universidad Carlos III de Madrid

Reporte de Data carving - Capture the Flag

Juan Diego Llano Miraval

Fecha: 18/05/2024

Procedure

The first step on this is to verify the content of the pcap file:

fore2.pcap

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	host	3.3.0	USB	64	GET_DESCRIPTOR Request DEVICE
2	0.001014	3.3.0	host	USB	82	GET_DESCRIPTOR Response DEVICE
3	0.001045	host	3.1.0	USB	64	GET_DESCRIPTOR Request DEVICE
4	0.001052	3.1.0	host	USB	82	GET_DESCRIPTOR Response DEVICE
5	0.475799	host	3.3.2	USBMS	95	SCSI: Test Unit Ready LUN: 0x00
6	0.476781	3.3.2	host	USB	64	URB_BULK out
7	0.476799	host	3.3.1	USB	64	URB_BULK in
8	0.476818	3.3.1	host	USBMS	77	
9	2.523755	host	3.3.2	USBMS	95	SCSI: Test Unit Ready LUN: 0x00
10	2.524758	3.3.2	host	USB	64	URB_BULK out
11	2.524785	host	3.3.1	USB	64	URB_BULK in
12	2.524801	3.3.1	host	USBMS	77	
13	4.571793	host	3.3.2	USBMS	95	SCSI: Test Unit Ready LUN: 0x00
14	4.572803	3.3.2	host	USB	64	URB_BULK out
15	4.572829	host	3.3.1	USB	64	URB_BULK in
16	4.572843	3.3.1	host	USBMS	77	
17	6.619740	host	3.3.2	USBMS	95	SCSI: Test Unit Ready LUN: 0x00
18	6.620726	3.3.2	host	USB	64	URB_BULK out
19	6.620748	host	3.3.1	USB	64	URB_BULK in
20	6.620761	3.3.1	host	USBMS	77	
21	8.667739	host	3.3.2	USBMS	95	SCSI: Test Unit Ready LUN: 0x00
22	8.668712	3.3.2	host	USB	64	URB_BULK out
23	8.668739	host	3.3.1	USB	64	URB_BULK in
24	8.668752	3.3.1	host	USBMS	77	
25	10.715842	host	3.3.2	USBMS	95	SCSI: Test Unit Ready LUN: 0x00
26	10.716859	3.3.2	host	USB	64	URB_BULK out
27	10.716888	host	3.3.1	USB	64	URB_BULK in
28	10.716903	3.3.1	host	USBMS	77	

Frame 1: 64 bytes on wire (512 bits), 64 bytes captured (512 bits)

USB URB

Setup Data

0000

80 64 d2

0010

2e 09 68

0020

28 00 00

0030

00 00 00

As we were mentioned, there are two ways to find the required data, one of them is by converting the data into binary. The first step for this is to use tshark to convert the data into hex:

```
(root@kali)-[/home/kali/Desktop]
# tshark -r fore2.pcap -x > payload.txt
Running as user "root" and group "root". This could be dangerous.

BULK OUT
(root@kali)-[/home/kali/Desktop]
# cat payload.txt
0000  80 64 d2 ec bc 8f ff ff 53 02 80 03 03 00 00 3c  .d.....S.....<
0010  2e 09 68 58 00 00 00 00 06 20 00 00 8d ff ff ff  ..hX.....
0020  28 00 00 00 00 00 00 00 00 80 06 00 01 00 00 28 00  (.....(
0030  00 00 00 00 00 00 00 00 00 00 02 00 00 00 00 00  .....

0000  80 64 d2 ec bc 8f ff ff 43 02 80 03 03 00 2d 00  .d.....C.....-
0010  2e 09 68 58 00 00 00 00 fc 23 00 00 00 00 00 00  ..hX.....#.....
0020  12 00 00 00 12 00 00 00 00 00 00 00 00 00 00 00  .....
```

We later created a small python code to transform the hex into binary:

```
import re

def hex_to_bin(input_file, output_file):

    with open(input_file, 'r') as f:

        lines = f.readlines()

    hex_data = []

    for line in lines:

        # Match hex data lines which are of the form: "0000 00 11 22 33 44 55 66
        77 88 99 aa bb cc dd ee ff"

        match = re.search(r'^[0-9a-fA-F]{4}\s+((?:[0-9a-fA-F]{2}\s+)+)', line)

        if match:

            hex_line = match.group(1).replace(' ', '')

            hex_data.append(hex_line)

    with open(output_file, 'wb') as f:

        for hex_line in hex_data:
```

```
f.write(bytes.fromhex(hex_line))

hex_to_bin('payload.txt', 'output.bin')
```

```
(root@kali)-[/home/kali/Desktop]
# python hex_to_bin.py

(root@kali)-[/home/kali/Desktop]
# cat out.bin

(root@kali)-[/home/kali/Desktop]
# cat output.bin
d* S<. hX +++++(+d* C+-. hX# Q +d* S<. hX++++(+d* C+-. hX"$ +d*
++S-. hX+b++++USBC`
JSBS`C+-. hX+f++++S+<-. hX+f++++
++++S-0 hX++++USBCa
JSBSaC+0 hX"++++S+<0 hX++++
++++S-2 hX++++USBCb
JSBSbC+2 hX++++S+<2 hX++++
++++S-4 hX++ +++++USBCc
JSBScC+4 hX++ +++++S+<4 hX3 +++++
++++S-6 hXaP
+++USBCd
++++C+6 hX.T
+++S+16 hX.T
```

We run the code and we get the binary of the pcap. After this we run binwalk with -e to extract any file inside.

```
(root@kali)-[/home/kali/Desktop/hextobin]
# binwalk -e output.bin --run-as=root
```

DECIMAL	HEXADECIMAL	DESCRIPTION
60659	0xECF3	PNG image, 460 x 130, 8-bit/color RGBA, interlaced

```
(root@kali)-[/home/kali/Desktop/hextobin]
# ls
output.bin
```

Unfortunately, binwalk was not able to extract the file, but we now know that there is a PNG file inside, and we also have the position. With the help of the `dd` tool we extracted the file:

```
(root@kali)-[/home/kali/Desktop/hextobin]
# dd if=output.bin of=extracted_image.png bs=1 skip=60659
85981+0 records in
85981+0 records out
85981 bytes (86 kB, 84 KiB) copied, 0.0833592 s, 1.0 MB/s
```

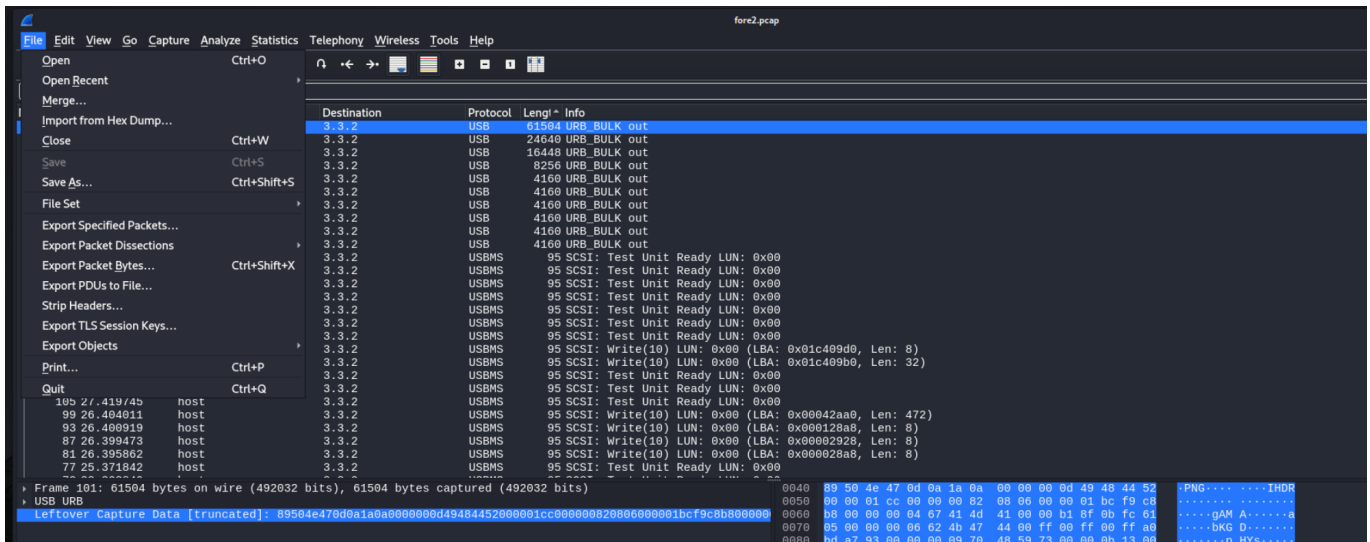
We exported the file as a png, and we checked again if the file indeed is a png:

```
(root@kali)-[/home/kali/Desktop/hextobin]
# file extracted_image.png
extracted_image.png: PNG image data, 460 x 130, 8-bit/color RGBA, interlaced
```

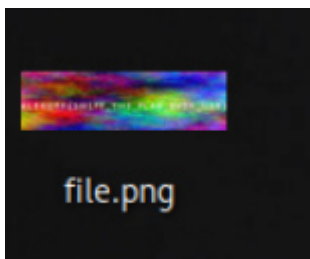
When we open this image we get the flag we were looking for:



for the second method, i opened the file on wireshark and filter trough length. This filter allows me to see the messages that have more information, and it didn't take too much effort to find a packet with a png header on the data. I exported the file from wireshark and i get the image we were looking for:



having selected the data in wireshark, we go to file, export packet bytes, and then we save the file. What we get is the image we were looking for:



The flag is: ALEXCTF{SN1FF_TH3_FL4G_OV3R_USB}