USB Evidence Forensics CTF Report
Cybersecurity Forensics Analyst: Apurv Nandgaonkar
Date: 21-11-2024

Executive Summary

This report outlines the forensic analysis of a suspicious USB image file found during a cyber-incident investigation. The goal was to uncover hidden flags within the files of the USB image . Through a detailed analysis of the USB image, two flags were successfully identified and decoded.

Objectives

- Analyze the provided USB image file.
- Locate hidden files containing sensitive information.
- Decode and extract any flags.
- Analyze the associated PCAP file for hidden network-based flags.

Methodology

The forensic investigation was performed in the following stages:

1. **Extraction of Files**: Unzipped and extracted the contents of the provided USB archive.
2. **Disk Image Analysis**: Mounted and examined the USB image for any hidden or suspicious files.
3. **Hexadecimal and Base64 Decoding**: Identified and decoded hex and Base64 strings within recovered files to find flags.
4. **PCAP File Analysis**: Inspected the PCAP file for Base64-encoded flags within packets.

Step-by-Step Process

Step 1: Extracting the Zip Archive

The first step involved extracting the contents of the file **usbevidence.zip** using the following command:

```
$ unzip usbevidence.zip
```

This resulted in the **usbevidence.7z** archive.

Step 2: Extracting the 7z Archive

The **usbevidence.7z** file was then extracted using the **7z** tool:

```
$ 7z x usbevidence.7z
```

This extraction produced the **usbevidence.001** file.

Step 3: Converting .001 to .img File

To convert the **usbevidence.001** file into a usable disk image, the following command was used:

```
$ cat usbevidence.001 > usbevidence.img
```

Step 4: Identifying the File Type

To analyze the type of the disk image, the **file** command was run:

```
$ file usbevidence.img
```

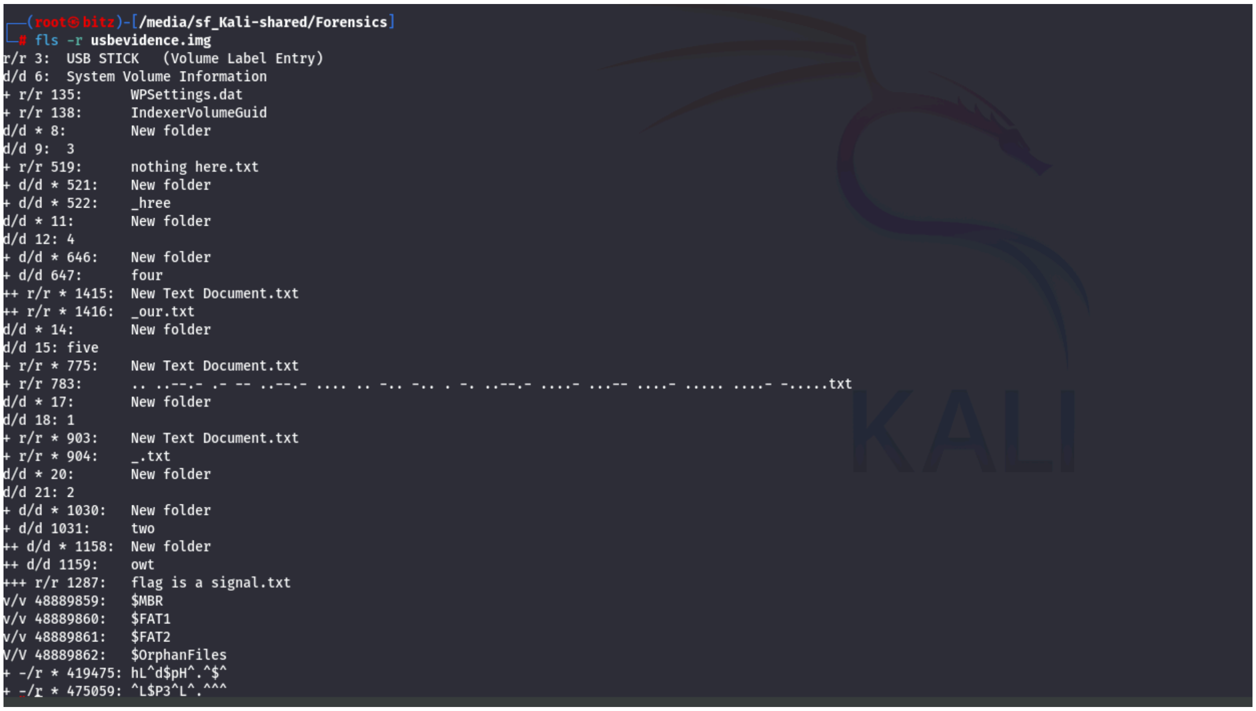
This revealed that the file was a DOS/MBR boot sector, confirming the presence of a FAT32 partition on the disk.

Step 5: Mounting the Disk Image

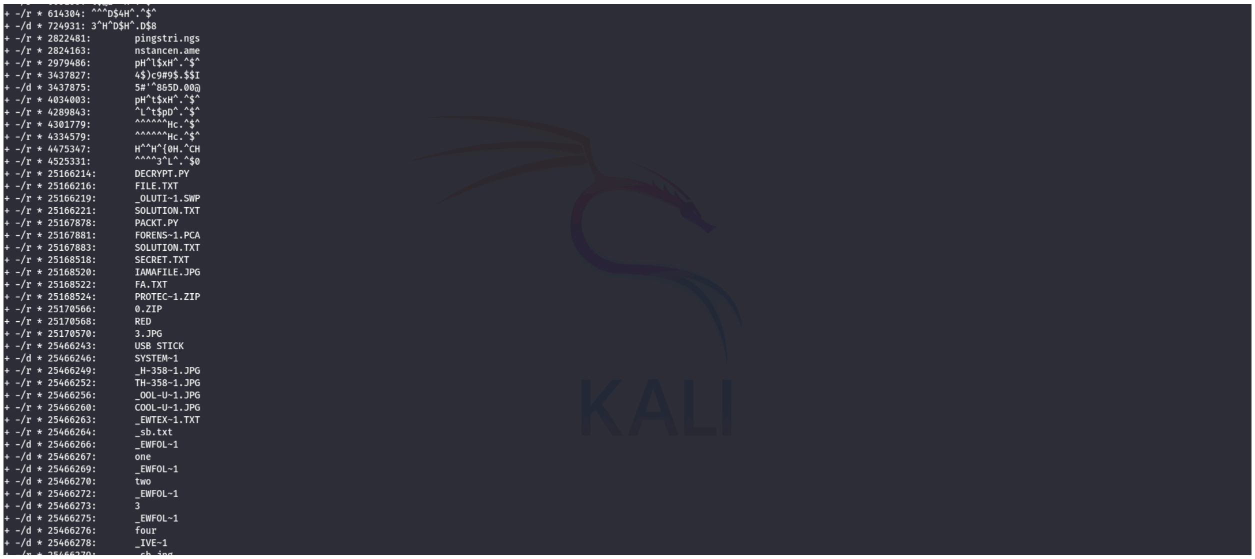
The image was then mounted to allow further inspection of the file system. Using the **SleuthKit** tools, the following command was executed to list files and directories within the image:

```
$ sleuthkit fls -r usbevidence.img
```

This outputted a list of files, including the **secret.txt** file, which was of interest.



```
┌──(root㉿bitz)-[/media/sf_Kali-shared/Forensics]
└─# fls -r usbevidence.img
r/r 3:  USB STICK   (Volume Label Entry)
d/d 6:  System Volume Information
+ r/r 135:      WPSettings.dat
+ r/r 138:      IndexerVolumeGuid
d/d * 8:        New folder
d/d 9:  3
+ r/r 519:      nothing here.txt
+ d/d * 521:    New folder
+ d/d * 522:    _hree
d/d * 11:       New folder
d/d 12: 4
+ d/d * 646:    New folder
+ d/d 647:      four
++ r/r * 1415:  New Text Document.txt
++ r/r * 1416:  _our.txt
d/d * 14:       New folder
d/d 15: five
+ r/r * 775:    New Text Document.txt
+ r/r 783:      .. ..--.- .- -- ..--.- .... .. -.. -.. . -. ..--.- ....- ...-- ....- ..... ....- -.....txt
d/d * 17:       New folder
d/d 18: 1
+ r/r * 903:    New Text Document.txt
+ r/r * 904:    _.txt
d/d * 20:       New folder
d/d 21: 2
+ d/d * 1030:   New folder
+ d/d 1031:     two
++ d/d * 1158:  New folder
++ d/d 1159:    owt
+++ r/r 1287:   flag is a signal.txt
v/v 48889859:   $MBR
v/v 48889860:   $FAT1
v/v 48889861:   $FAT2
V/V 48889862:   $OrphanFiles
+ -/r * 419475: hL^d$pH^.^$^
+ -/r * 475059: ^L$P3^L^.^^^
```



```
+ -/r * 614304: ^^^D$4H^.^$^
+ -/d * 724931: 3^H^D$H^.D$8
+ -/r * 2822481:        pingstri.ngs
+ -/r * 2824163:        nstancen.ame
+ -/r * 2979486:        pH^l$xH^.^$^
+ -/r * 3437827:        4$)c9#9$.$$I
+ -/d * 3437875:        5#'^8&5D.00@
+ -/r * 4034003:        pH^t$xH^.^$^
+ -/r * 4289843:        ^L^t$pD^.^$^
+ -/r * 4301779:        ^^^^^^Hc.^$^
+ -/r * 4334579:        ^^^^^^Hc.^$^
+ -/r * 4475347:        H^^H^{0H.^CH
+ -/r * 4525331:        ^^^^3^L^.^$0
+ -/r * 25166214:       DECRYPT.PY
+ -/r * 25166216:       FILE.TXT
+ -/r * 25166219:       _OLUTI~1.SWP
+ -/r * 25166221:       SOLUTION.TXT
+ -/r * 25167878:       PACKT.PY
+ -/r * 25167881:       FORENS~1.PCA
+ -/r * 25167883:       SOLUTION.TXT
+ -/r * 25168518:       SECRET.TXT
+ -/r * 25168520:       IAMAFILE.JPG
+ -/r * 25168522:       FA.TXT
+ -/r * 25168524:       PROTEC~1.ZIP
+ -/r * 25170566:       0.ZIP
+ -/r * 25170568:       RED
+ -/r * 25170570:       3.JPG
+ -/r * 25466243:       USB STICK
+ -/d * 25466246:       SYSTEM~1
+ -/r * 25466249:       _H-358~1.JPG
+ -/r * 25466252:       TH-358~1.JPG
+ -/r * 25466256:       _OOL-U~1.JPG
+ -/r * 25466260:       COOL-U~1.JPG
+ -/r * 25466263:       _EWTEX~1.TXT
+ -/r * 25466264:       _sb.txt
+ -/d * 25466266:       _EWFOL~1
+ -/d * 25466267:       one
+ -/d * 25466269:       _EWFOL~1
+ -/d * 25466270:       two
+ -/d * 25466272:       _EWFOL~1
+ -/d * 25466273:       3
+ -/d * 25466275:       _EWFOL~1
+ -/d * 25466276:       four
+ -/d * 25466278:       _IVE~1
+ -/r * 25466279:       _sb.jpg
```

Step 6: Recovering and Analyzing secret.txt

Using **icat** to recover the **secret.txt** file:

```
$ sleuthkit icat usbevidence.img [25168518] > secret.txt
```

Inside the file, a hexadecimal string was found:

```
666c61677b7930755f6730745f6d335f343635377d
```

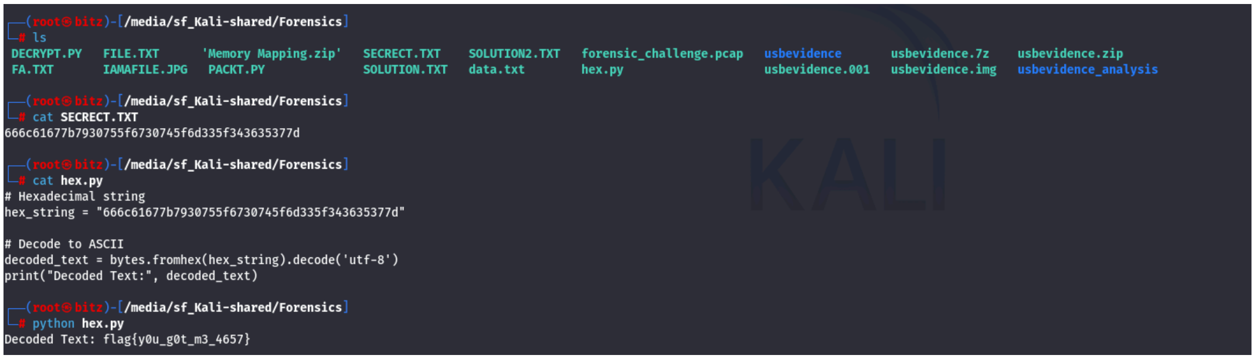
Step 7: Decoding the Hex String

The hexadecimal string was decoded using a Python script:

```
import binascii

hex_string = "666c61677b7930755f6730745f6d335f343635377d"
ascii_string = binascii.unhexlify(hex_string).decode('utf-8')

print(ascii_string)
```

The decoded string revealed the flag:

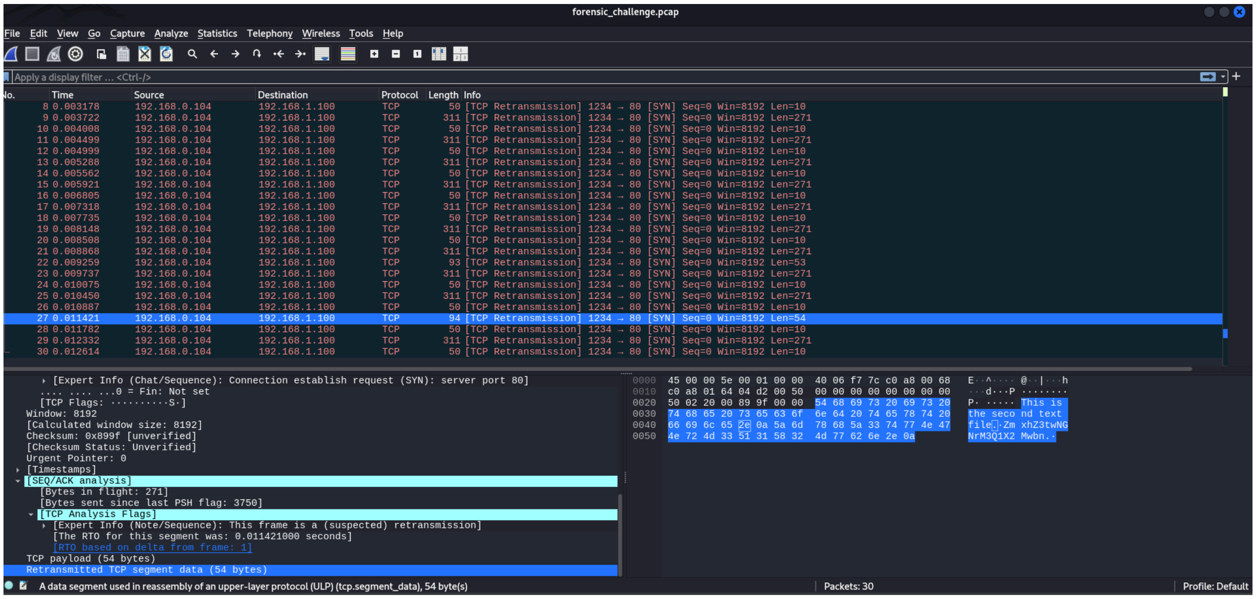
```
Flag{y0u_g0t_m3_4657}
```



```
┌──(root㉿bitz)-[/media/sf_Kali-shared/Forensics]
└─# ls
DECRYPT.PY   FILE.TXT      'Memory Mapping.zip'   SECRECT.TXT    SOLUTION2.TXT   forensic_challenge.pcap   usbevidence       usbevidence.7z    usbevidence.zip
FA.TXT       IAMAFILE.JPG   PACKT.PY              SOLUTION.TXT   data.txt        hex.py                    usbevidence.001   usbevidence.img   usbevidence_analysis

┌──(root㉿bitz)-[/media/sf_Kali-shared/Forensics]
└─# cat SECRECT.TXT
666c61677b7930755f6730745f6d335f343635377d

┌──(root㉿bitz)-[/media/sf_Kali-shared/Forensics]
└─# cat hex.py
# Hexadecimal string
hex_string = "666c61677b7930755f6730745f6d335f343635377d"

# Decode to ASCII
decoded_text = bytes.fromhex(hex_string).decode('utf-8')
print("Decoded Text:", decoded_text)

┌──(root㉿bitz)-[/media/sf_Kali-shared/Forensics]
└─# python hex.py
Decoded Text: flag{y0u_g0t_m3_4657}
```

Step 8: Analyzing the PCAP File

A PCAP file containing 30 packets was provided. Using **Wireshark**, packets 22 and 27 were identified as containing Base64-encoded data.


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

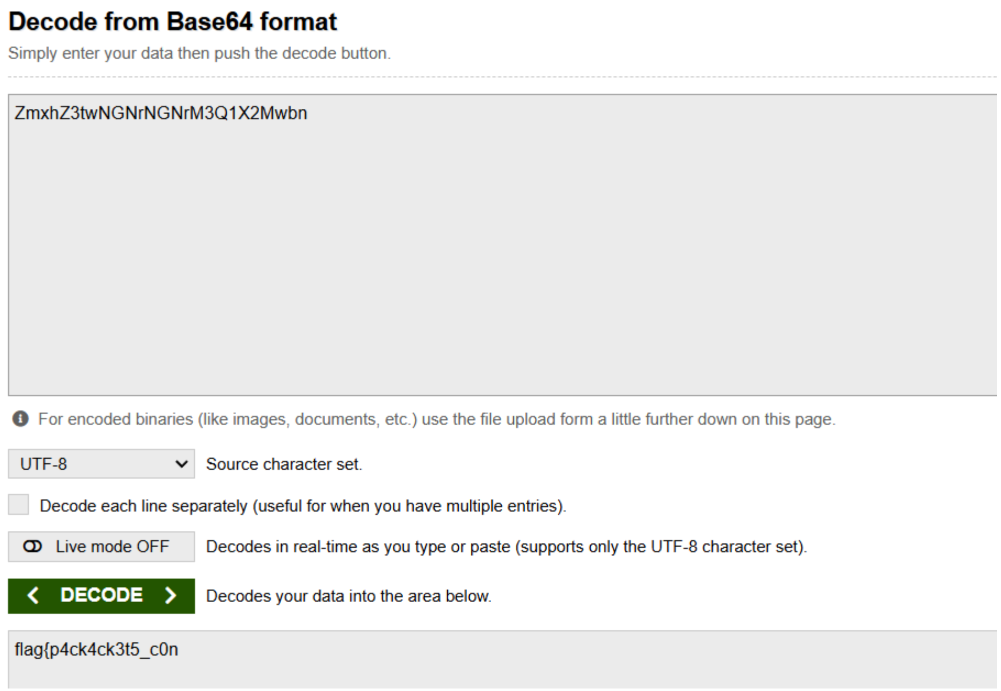
```
No.   Time      Source         Destination    Protocol Length Info
    8 0.003178  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
    9 0.003722  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   10 0.004008  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   11 0.004499  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   12 0.004999  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   13 0.005288  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   14 0.005562  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   15 0.005921  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   16 0.006805  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   17 0.007318  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   18 0.007735  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   19 0.008148  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   20 0.008508  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   21 0.008868  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   22 0.009259  192.168.0.104  192.168.1.100  TCP       93 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=53
   23 0.009737  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   24 0.010075  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   25 0.010450  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   26 0.010887  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   27 0.011421  192.168.0.104  192.168.1.100  TCP       94 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=54
   28 0.011782  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
   29 0.012332  192.168.0.104  192.168.1.100  TCP      311 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=271
   30 0.012614  192.168.0.104  192.168.1.100  TCP       50 [TCP Retransmission] 1234 → 80 [SYN] Seq=0 Win=8192 Len=10
```

```
      › [Expert Info (Chat/Sequence): Connection establish request (SYN): server port 80]
    .... .... ...0 = Fin: Not set
    [TCP Flags: ··········S·]
  Window: 8192
  [Calculated window size: 8192]
  Checksum: 0x899f [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
› [Timestamps]
⌄ [SEQ/ACK analysis]
    [Bytes in flight: 271]
    [Bytes sent since last PSH flag: 3750]
  ⌄ [TCP Analysis Flags]
    › [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
      [The RTO for this segment was: 0.011421000 seconds]
      [RTO based on delta from frame: 1]
  TCP payload (54 bytes)
  Retransmitted TCP segment data (54 bytes)
```

```
0000  45 00 00 5e 00 01 00 00  40 06 f7 7c c0 a8 00 68   E··^···· @··|···h
0010  c0 a8 01 64 04 d2 00 50  00 00 00 00 00 00 00 00   ···d···P ········
0020  50 02 20 00 89 9f 00 00  54 68 69 73 20 69 73 20   P· ····· This is 
0030  74 68 65 20 73 65 63 6f  6e 64 20 74 65 78 74 20   the seco nd text 
0040  66 69 6c 65 2e 0a 5a 6d  78 68 5a 33 74 77 4e 47   file.·Zm xhZ3twNG
0050  4e 72 4d 33 51 31 58 32  4d 77 62 6e 2e 0a         NrM3Q1X2 Mwbn.·
```

A data segment used in reassembly of an upper-layer protocol (ULP) (tcp.segment_data), 54 byte(s) | Packets: 30 | Profile: Default

Step 9: Decoding Base64 Strings



Decode from Base64 format

Simply enter your data then push the decode button.

```
ZmxhZ3twNGNrNGNrM3Q1X2Mwbn
```

For encoded binaries (like images, documents, etc.) use the file upload form a little further down on this page.

UTF-8 Source character set.

□ Decode each line separately (useful for when you have multiple entries).

Live mode OFF Decodes in real-time as you type or paste (supports only the UTF-8 character set).

DECODE Decodes your data into the area below.

```
flag{p4ck4ck3t5_c0n
```

The decoded flags from the PCAP file were:

- **Packet 27**: `flag{p4ck4ck3t5_c0n}`

Results

The forensic analysis led to the successful extraction of the following flags:

1. `flag{y0u_g0t_m3_4657}`
2. `flag{p4ck4ck3t5_c0n}`

The first flag was found within **secret.txt** in the USB image, and the second flag was discovered within the PCAP file through Base64 decoding of packets 22 and 27.

Conclusion

This forensic investigation demonstrates the utility of examining both disk images and network traffic to uncover hidden information. By utilizing file system analysis, hexadecimal decoding, and Base64 decoding techniques, the hidden flags were extracted successfully.

References

- Wireshark documentation
- SleuthKit documentation