

MISC 达芬奇密码

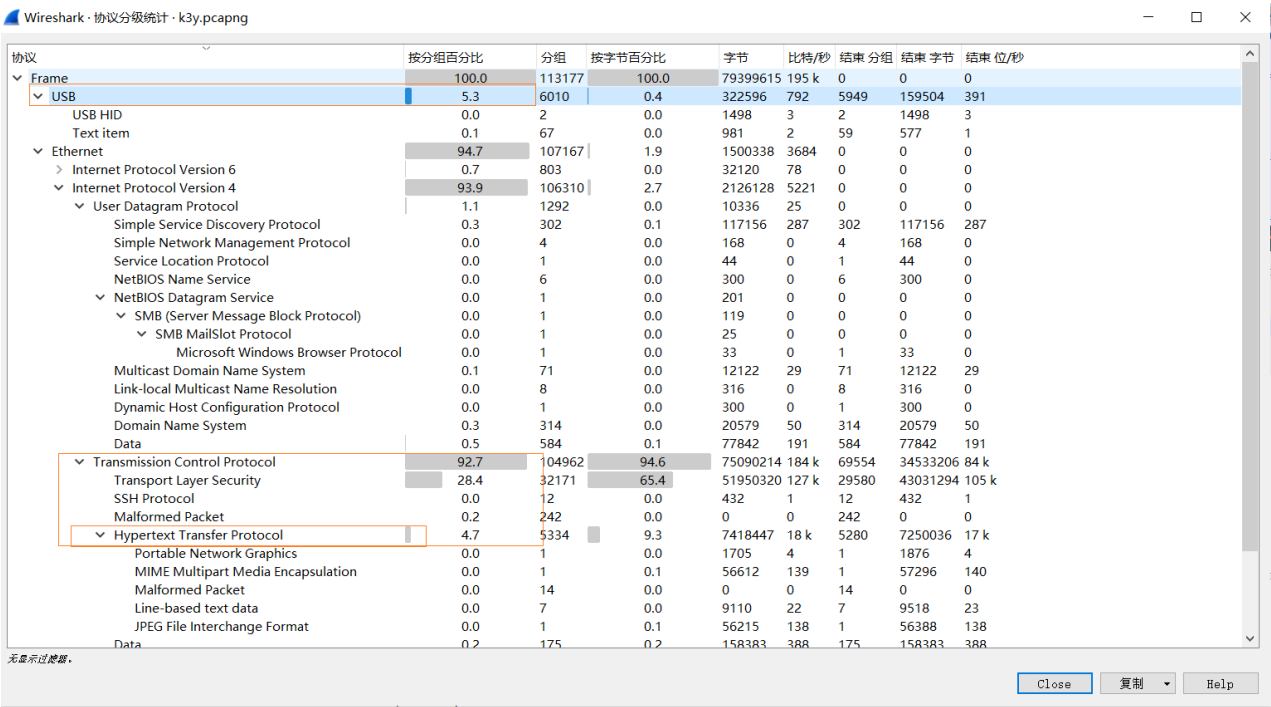
题目描述

达芬奇偷偷把key画了下来，你能找到key然后解开密码吗？

解题思路

- 1. 附件有一个流量包，有个flagtxt是unicode，结合题意应该是从流量包中找到线索去解决flagtxt中内容
- 2. 分析流量

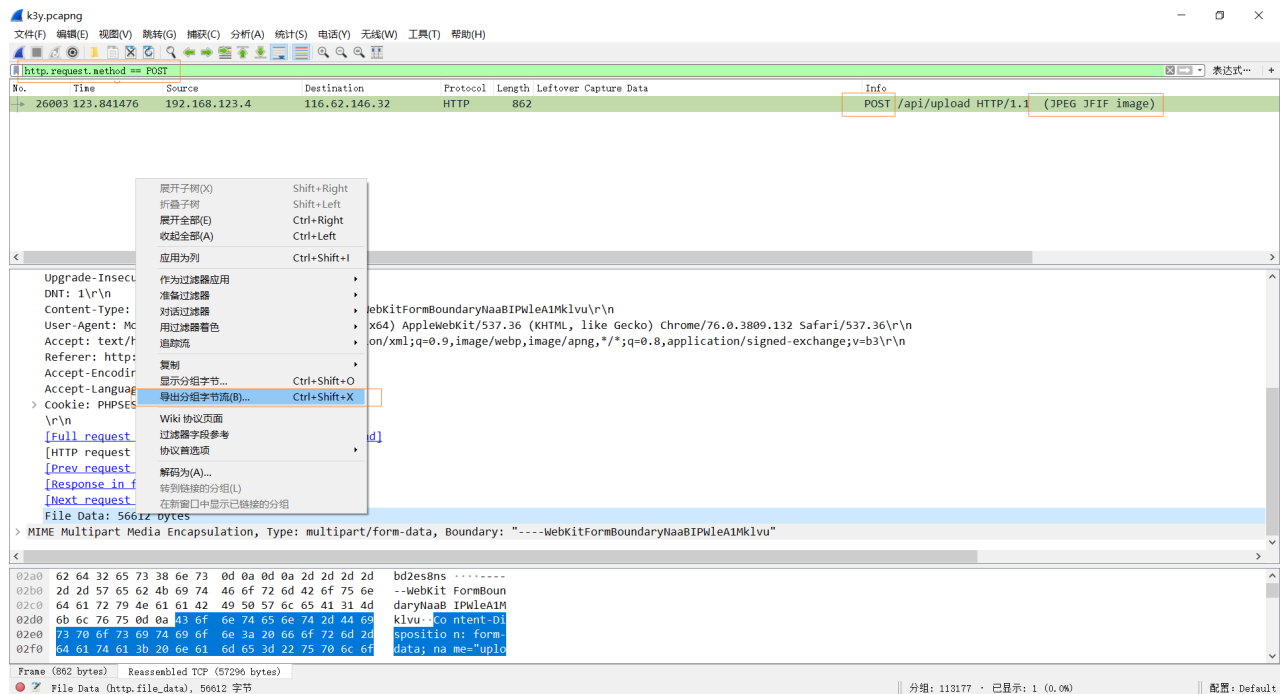
大致看流量包的协议组成，会话地址等等



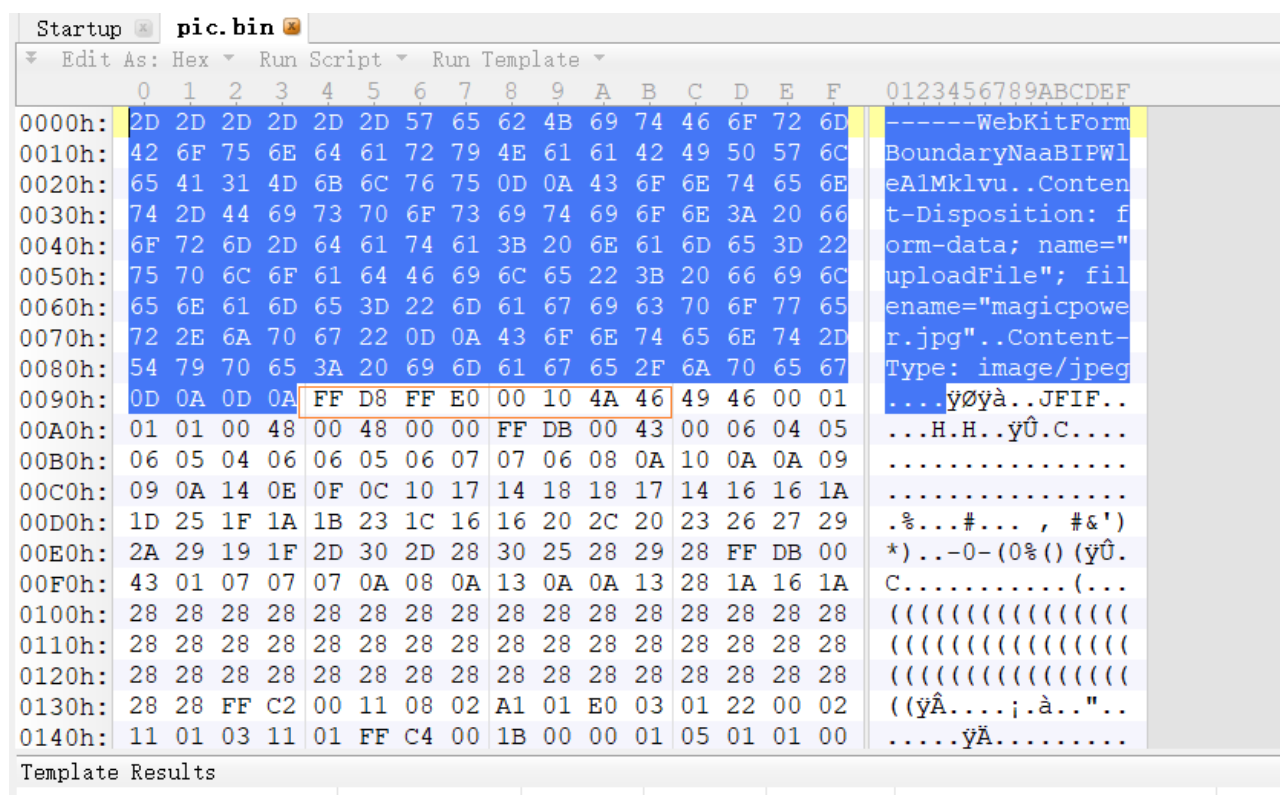
可以观察到既有usb流量又有网络流量

- 3. 分析网络流量

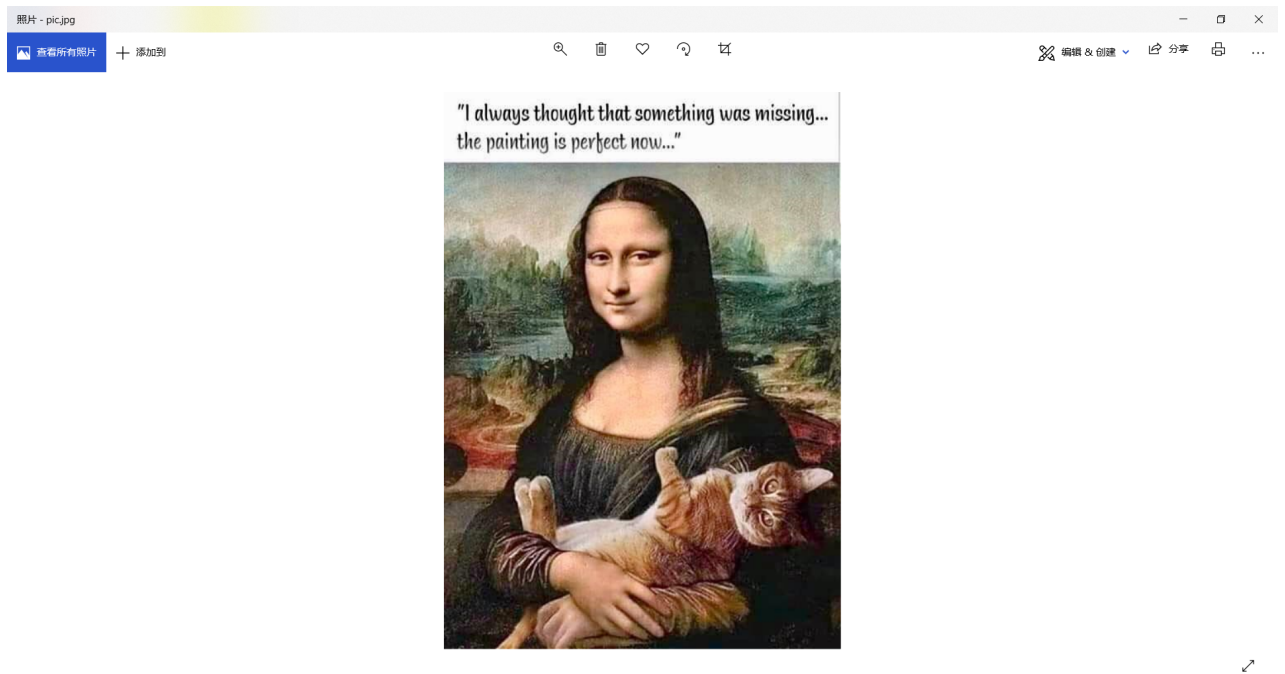
筛选http流量，可以从中找到一个post包并且可以发现post的是jpg文件（可通过http.request.method筛选）



如图导出分组字节流，在010editor中打开删除头部无用字节只保留jpg字节



另存为jpg



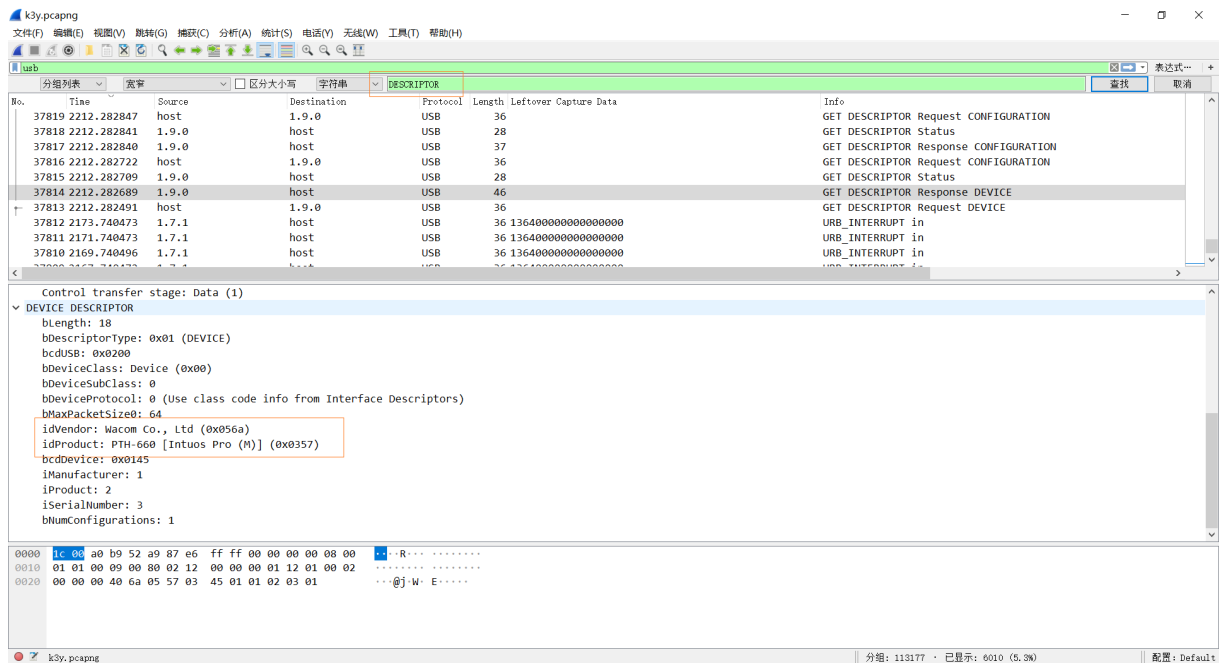
可以用各种工具检查这张图片，但是并不会有什么结果🐼（烟雾弹）

4. 分析usb流量

1. 判断usb设备

usb设备建立连接时会发送包含设备信息的包，搜索关键字DESCRIPTOR

(详细可见usb协议https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf 3.6小节)



可以找到一个wacom pth660的设备，搜索后可以发现是个数位板，该设备source 1.11.1，另外还有一个设备1.9.1可能是鼠标或键盘

2. 过滤有效流量

usb流量的数据域是Leftover Capture Data，数位板设备src 1.11.1，筛选后可以判断frame length为54时 leftover cap data域携带信息更多，更可能是有效流量

全部筛选条件 `(usb.src == "1.11.1") && (frame.len == 54)` 并导出筛选后的分组test.pcapng

3. 分析数据域

usb协议https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf 中写了数位板的数据基本格式，并且给出了几个数位板数据格式示例

Figure 17: Example Digitizer 2-Button Stylus Input Report136

Figure 18: Example Digitizer 16-Button Puck Input Report136

Figure 19: Example Digitizer Pressure Stylus Input Report.....136

wacom pth660经搜索可以发现是一块带压感的专业绘画板，所以可以参考figure19

Figure 19: Example Digitizer Pressure Stylus Input Report

Byte	Bit							
	7	6	5	4	3	2	1	0
0	ReportID = 3 (indicating pressure stylus report)							
1	X Coordinate Bits 0-7							
2	X Coordinate Bits 8-15							
3	Y Coordinate Bits 0-7							
4	Y Coordinate Bits 8-15							
5	In Range	Barrel Switch	Unused					
6	Pressure Bits 0-7							

需要注意的是，x,y坐标的储存是**按照小端序**，另外**控制筛选压力值**可以使作图结果更精准

观察提取出的leftover capture data数据可以猜测找到坐标存储位置，红框为坐标高位bit，同一时间内变化小于绿框（低位bit）变化率，橙框变化猜测为压力值，笔离开画板时压力变为0

```

10:61:d9:5b:00:e6:45:00:08:18:05:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:da:5b:00:e6:45:00:e7:17:05:f7:00:00:00:00:07:84:30:80:94:42:08:10:00:42:08
10:61:da:5b:00:e7:45:00:c9:17:05:f7:00:00:00:00:07:84:30:80:94:42:08:10:00:42:08
10:61:db:5b:00:e8:45:00:ab:17:05:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:dc:5b:00:e9:45:00:8a:17:05:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:dd:5b:00:ea:45:00:69:17:05:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:dd:5b:00:eb:45:00:42:17:05:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:de:5b:00:ec:45:00:0f:17:05:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:de:5b:00:ee:45:00:cd:16:05:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:de:5b:00:ef:45:00:79:16:05:f7:00:00:00:00:09:84:30:80:94:42:08:10:00:42:08
10:61:dd:5b:00:f1:45:00:13:16:06:f7:00:00:00:00:09:84:30:80:94:42:08:10:00:42:08
10:61:db:5b:00:f2:45:00:95:15:06:f7:00:00:00:00:08:84:30:80:94:42:08:10:00:42:08
10:61:d9:5b:00:f4:45:00:f3:14:06:f7:00:00:00:00:09:84:30:80:94:42:08:10:00:42:08
10:61:d7:5b:00:f7:45:00:2a:14:06:f7:00:00:00:00:09:84:30:80:94:42:08:10:00:42:08
10:61:d4:5b:00:fb:45:00:40:13:06:f7:00:00:00:00:09:84:30:80:94:42:08:10:00:42:08
10:61:d0:5b:00:ff:45:00:23:12:06:f8:00:00:00:00:09:84:30:80:94:42:08:10:00:42:08
10:61:cc:5b:00:05:46:00:c1:10:06:f8:00:00:00:00:0a:84:30:80:94:42:08:10:00:42:08
10:61:c7:5b:00:0e:46:00:cf:0e:06:f8:00:00:00:00:0a:84:30:80:94:42:08:10:00:42:08
10:61:c1:5b:00:19:46:00:1a:0c:06:f8:00:00:00:00:0b:84:30:80:94:42:08:10:00:42:08
10:61:b8:5b:00:28:46:00:9a:07:06:f8:00:00:00:00:0b:84:30:80:94:42:08:10:00:42:08
10:61:ad:5b:00:3a:46:00:a4:01:06:f8:00:00:00:00:0d:84:30:80:94:42:08:10:00:42:08
10:60:ad:5b:00:3a:46:00:00:00:06:f8:00:00:00:00:0d:84:30:80:94:42:08:10:00:42:08
10:60:95:5b:00:67:46:00:00:00:07:f8:00:00:00:00:0f:84:30:80:94:42:08:10:00:42:08
10:60:91:5b:00:70:46:00:00:00:07:f8:00:00:00:00:12:84:30:80:94:42:08:10:00:42:08
10:60:8d:5b:00:77:46:00:00:00:07:f8:00:00:00:00:14:84:30:80:94:42:08:10:00:42:08
10:60:8a:5b:00:7b:46:00:00:00:07:f8:00:00:00:00:17:84:30:80:94:42:08:10:00:42:08
10:60:82:5b:00:83:46:00:00:00:07:f7:00:00:00:00:1a:84:30:80:94:42:08:10:00:42:08
10:60:7b:5b:00:87:46:00:00:00:08:f7:00:00:00:00:1e:84:30:80:94:42:08:10:00:42:08
10:60:75:5b:00:88:46:00:00:00:08:f7:00:00:00:00:1f:84:30:80:94:42:08:10:00:42:08
10:60:72:5b:00:87:46:00:00:00:08:f7:00:00:00:00:22:84:30:80:94:42:08:10:00:42:08
10:60:71:5b:00:84:46:00:00:00:08:f7:00:00:00:00:25:84:30:80:94:42:08:10:00:42:08
10:60:73:5b:00:7d:46:00:00:00:09:f7:00:00:00:00:27:84:30:80:94:42:08:10:00:42:08

```

4. 脚本画图

根据数据画图，脚本如下（魔改了wangyihang大佬的鼠标流量脚本

用脚本处理筛选后的分组

```
python UsbDigitizerHacker.py test.pcapng
```

```

# coding:utf-8

import sys
import os
import numpy as np
import matplotlib.pyplot as plt

mousePositionX = 0
mousePositionY = 0

X = []
Y = []

DataFileName = "test.txt"
data = []

def main():
    global mousePositionX
    global mousePositionY
    # check argv
    if len(sys.argv) == 1:

```

```

        print "Usage : "
        print "          python UsbDigitizerHacker.py data.pcap [Conditions used
to sort]"
        print "Tips : "
        print "          To use this python2 script , you must install the
numpy,matplotlib first."
        print "          You can use `sudo pip install matplotlib numpy` to
install it"
        exit(1)

# get argv
pcapFilePath = sys.argv[1]
print pcapFilePath
# get data of pcap
if len(sys.argv)==2:
    command = "tshark -r '%s' -T fields -e usb.capdata > %s" % (
        pcapFilePath, DataFileName)
    print command
    os.system(command)
if len(sys.argv)==3:
    Conditions=sys.argv[2]
    command = "tshark -r '%s' -T fields -e usb.capdata -Y '%s' > %s" % (
        pcapFilePath,Conditions, DataFileName)
    print command
    os.system(command)

with open(DataFileName, "rb") as f:
    flag=1
    for line in f:
        if line[24:26] != "00": #根据压力值筛选, 使画图结果更精准
            print line
            data.append(line[0:-1])

#x,y坐标 小端序
for line in data:
    x0=int(line[6:8],16)
    x1=int(line[9:11],16)
    x=x0+x1*256
    y0=int(line[15:17],16)
    y1=int(line[18:20],16)
    y=y0+y1*256
    X.append(x)
    Y.append(-y)

#draw
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.set_title('%s' % (pcapFilePath))
ax1.scatter(X, Y, c='r', marker='o')
plt.savefig("out.png")
plt.show()

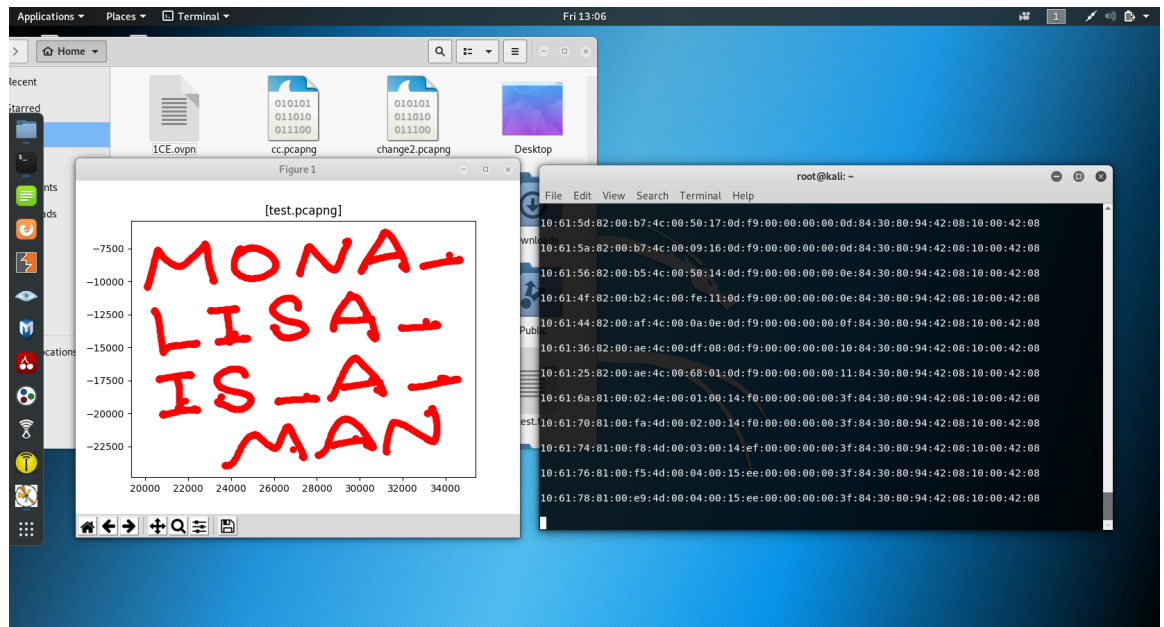
```



```
#clean temp data
os.system("rm ./%" % (DataFileName))

if __name__ == "__main__":
    main()
```

结果



如题目描述所说的key是 `MONA_LISA_IS_A_MAN`

4. 分析 flag.txt

看就知道是unicode

```
U+1F643U+1F4B5U+1F33FU+1F3A4U+1F6AAU+1F30FU+1F40EU+1F94BU+1F6ABU+1F606U+1F383U+1F9
93U+2709U+1F33FU+1F4C2U+2603U+1F449U+1F6E9U+2705U+1F385U+2328U+1F30FU+1F6E9U+1F6A8
U+1F923U+1F4A7U+1F383U+1F34DU+1F601U+2139U+1F4C2U+1F6ABU+1F463U+1F600U+1F463U+1F64
3U+1F3A4U+2328U+1F601U+1F923U+1F3A4U+1F579U+1F451U+1F6AAU+1F374U+1F579U+1F607U+1F3
74U+1F40EU+2705U+2709U+1F30FU+23E9U+1F40DU+1F6A8U+2600U+1F607U+1F3F9U+1F441U+1F463
U+2709U+1F30AU+1F6A8U+2716
```

转换(<https://r12a.github.io/app-conversion/>)可以发现都是emoji





搜索emoji cipher可以发现一个叫emoji-aes的东西...

<https://aghorler.github.io/emoji-aes/>

(其实有好几个与emoji加密相关的，但只有这个需要以一个字符串作为key)

message使用emoji串，key使用 MONA_LISA_IS_A_MAN

Decrypt

To decrypt, select the agreed rotation (if custom), enter the emoji-aes string, and then the pre-shared encryption key.

Advanced

Message

RoarCTF{wm-m0de3n_dav1chi}

Key

Decrypt

Decrypted!

得到RoarCTF{wm-m0de3n_dav1chi}