攻防世界———MISC 高手区题解

原创

Captain Hammer 于 2019-10-14 18:58:18 发布 13024 收藏 39
分类专栏： CTF题解

本文链接：https://blog.csdn.net/vhkjhwbs/article/details/100775409

目录

1，base64stego

很神奇的base 64 加密 （在base 64 的密文中加密，还不影响原明文）

（转载于： https://www.tr0y.wang/2017/06/14/Base64steg/ 做了一定的修改）

（简要原理，ascii码是用8位二进制表示一个字符的，而base64是用6位二进制表示一个字符，将明文字符转化为二进制后再每6位划分成 一个 “字节”，然后将每个字节转化为一个字符，就变成了base64密文，而在base64的密文中加密是利用，每一段密文的最后4位二进制是不影响明文的，可以将flag转化为二进制后拆分隐藏在每一段的最后4位二进制中）

复习一下 Base64 吧

BASE64 是一种编码方式, 是一种可逆的编码方式.
编码后的数据是一个字符串, 包含的字符为: A-Za-z0-9+/
共 64 个字符：26 + 26 + 10 + 1 + 1 = 64
其实是 65 个字符, “=”是填充字符.

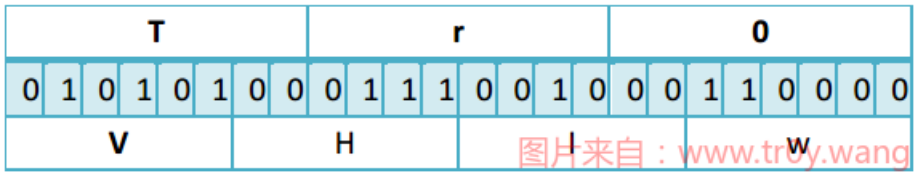
64 个字符需要 6 位二进制来表示, 表示成数值为 0~63.

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

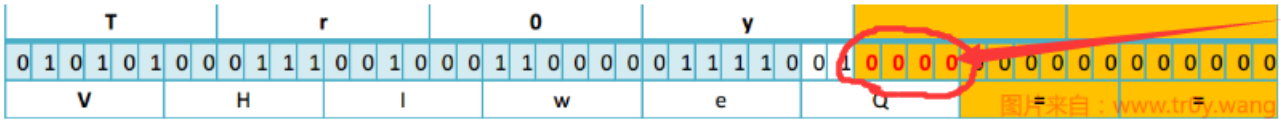
这样, 长度为 3 个字节的数据经过 Base64 编码后就变为 4 个字节

编码

比如, 字符串”Tr0”经过 Base64 编码后变为”VHlw”



上面说的字符串长度为 3 个字节的数据位数是 8x3=24, 可以精确地分成 6x4.
如果字节数不是 3 的倍数, 则位数就不是 6 的倍数, 那么就不能精确地划分成 6 位的块.
此时, 需在原数据二进制值后面添加零, 使其字节数是 6 的倍数.
然后, 在编码后的字符串后面添加 1 个或 2 个等号”=”, 表示所添加的零值字节数.
比如, 字符串”Tr0y”经过 Base64 编码后变为”VHlweQ==”



橙色底纹就是添加的 0.
这是 Base64 编码的方式.

解码

解码就是编码的逆过程.

1. 把 Base64 字符串去掉等号, 转为二进制数(VHlweQ== -> VHlweQ -> 010101000111001000110000011110010000).
2. 从左到右, 8 个位一组, 多余位的扔掉, 转为对应的 ASCII 码(01010100 01110010 00110000 01111001 0000 -> 扔掉最后 4 位 -> 01010100 01110010 00110000 01111001 -> Tr0y)

base64密文中隐写原理

注意红色的 0, 我们在解码的时候将其丢弃了, 所以这里的值不会影响解码. 所以我们可以在这进行隐写.
为什么等号的那部分 0 不能用于隐写? 因为修改那里的二进制值会导致等号数量变化, 解码的第 1 步会受影响. 自然也就破坏了源字符串.
而红色部分的 0 是作为最后一个字符二进制的组成部分, 还原时只用到了最后一个字符二进制的前部分, 后面的部分就不会影响还原.
唯一的影响就是最后一个字符会变化. 如下图

如果你直接解密'VHlweQ=='与'VHlweR==', 得到的结果都是'Tr0y'.

当然, 一行 base64 顶多能有 2 个等号, 也就是有 2*2 位的可隐写位. 所以我们得弄很多行, 才能隐藏一个字符串, 这也是为什么题目给了一大段 base64 的原因.
接下来, 把要隐藏的 flag 转为 8 位二进制, 塞进去就行了.

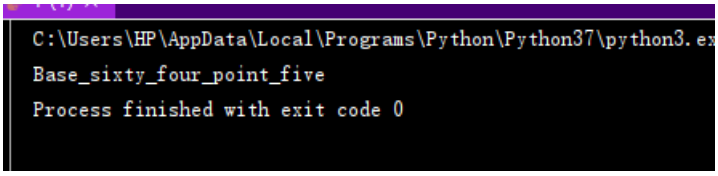
加密：

```
转载自：https://www.tr0y.wang/2017/06/14/Base64steg/
import base64
flag = 'Tr0y{Base64isF4n}' #flag
bin_str = ''.join([bin(ord(c)).replace('0b', '').zfill(8) for c in flag])
base64chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
with open('0.txt', 'rb') as f0, open('1.txt', 'wb') as f1: #'0.txt'是明文, '1.txt'用于存放隐写后的 base64
    for line in f0.readlines():
        rowstr = base64.b64encode(line.replace('\n', ''))
        equalnum = rowstr.count('=')
        if equalnum and len(bin_str):
            offset = int('0b'+bin_str[:equalnum * 2], 2)
            char = rowstr[len(rowstr) - equalnum - 1]
            rowstr = rowstr.replace(char, base64chars[base64chars.index(char) + offset])
            bin_str = bin_str[equalnum*2:]
        f1.write(rowstr + '\n')
```

解密：

```
import base64

b64chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
with open('stego.txt', 'rb') as f:   #stego.txt 为在base64密文中加密后的密文
    flag = ''
    bin_str = ''
    for line in f.readlines():
        stegb64 = str(line, "utf-8").strip("\n")
        rowb64 = str(base64.b64encode(base64.b64decode(stegb64)), "utf-8").strip("\n")
        offset = abs(b64chars.index(stegb64.replace('=', '')[-1]) - b64chars.index(rowb64.replace('=', '')[
        equalnum = stegb64.count('=')  # no equalnum no offset
        if equalnum:
            bin_str += bin(offset)[2:].zfill(equalnum * 2)
for i in range(0, len(bin_str), 8):
    print(chr(int(bin_str[i:i + 8], 2)),end='')
```

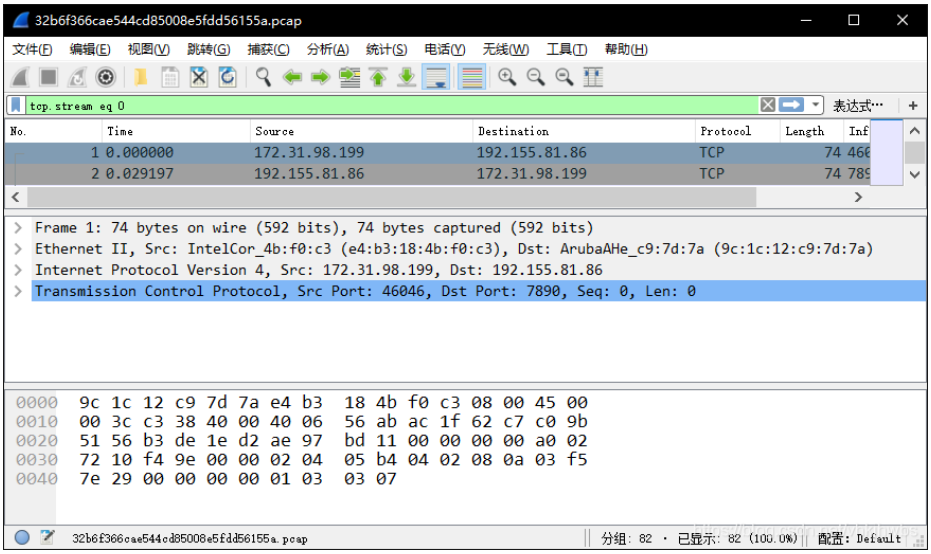
给出加密脚本执行结果

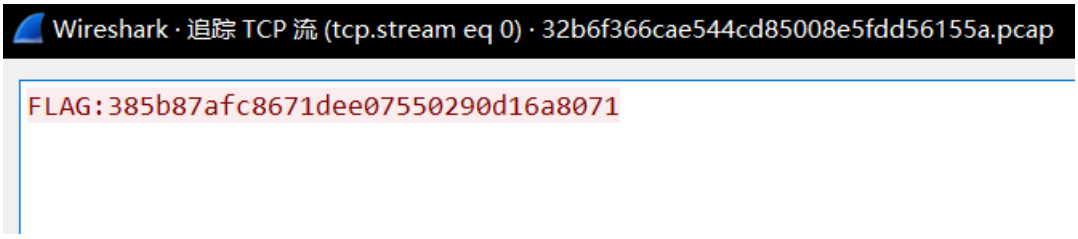


flag{Base_sixty_four_point_five}

2，easycap

下载下来是一个截取的流量包，用wireshark打开，先追踪流，没想到直接得到了flag：

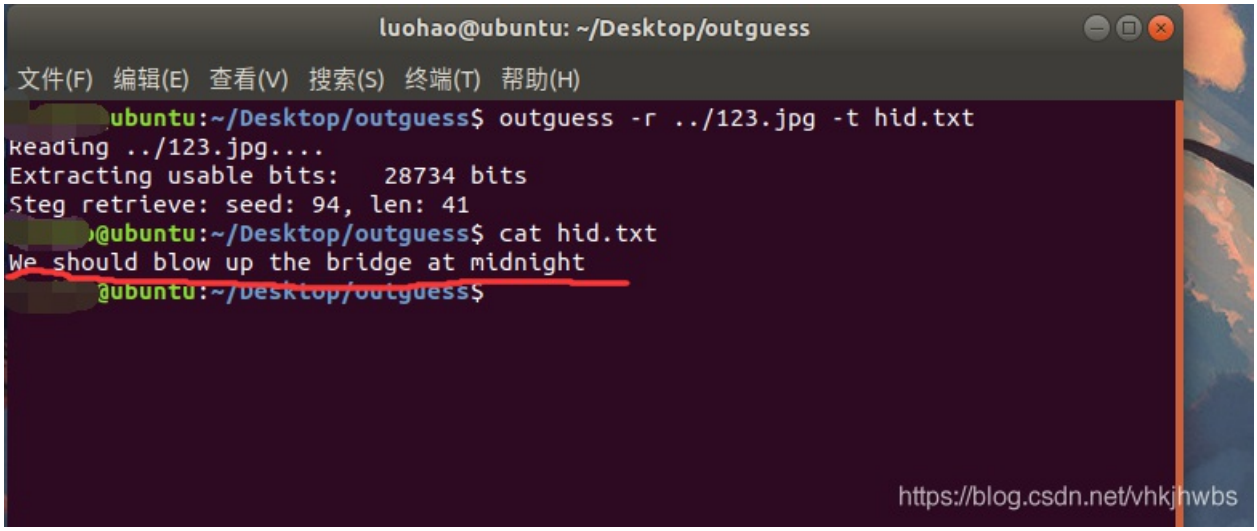




3，Avatar

是一张图片，先用formost分离一下，没有什么东西，用stegsolve看一下也没发现什么，再放进winhex中看一下，依旧没收获，

再试其他工具，……在用outguess时分离出来了隐藏信息：



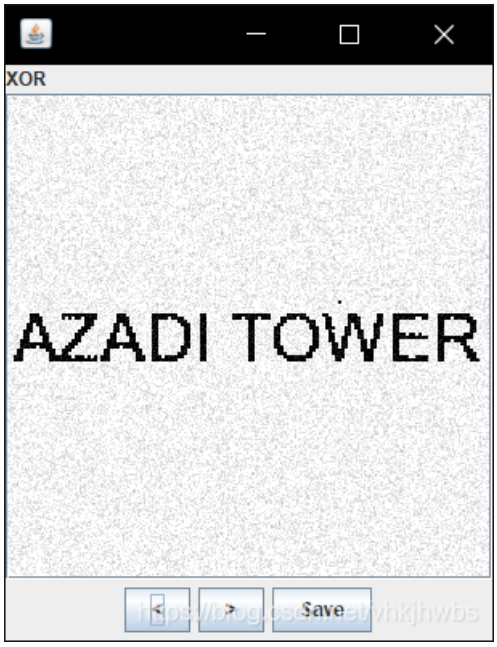
We should blow up the bridge at midnight
直接提交

4，What-is-this

解压文件，是一个没有后缀的文件，放进winhex中审查一下，发现有几个文件名，目测这是一个压缩包，把后缀给为zip后解压，

两张图片，正常思路：1，图片拼接 2，盲水印 3，各有一部分flag

先试一下 图片拼接，用stegsolve把两张图片合成一下：



直接提交

5，签到题

（本以为很简单，实际……）

将Z2dRQGdRMWZxaDBvaHRqcHRfc3d7Z2ZoZ3MjfQ== base64解密一下：ggQ@gQ1fqh0ohtjpt_sw{gfhgs#}

你以为这就结束了？提交错误

猜测{}的位置不对，进行一下栅栏密码：ggQ@gQ1fqh0ohtjpt_sw{gfhgs#}

2字一栏：gQg1q0hjts{fg#g@Qfhotp_wghs}
4字一栏：ggqht{ggQht_gsQ10jsf#@fopwh}
7字一栏：gfjggqpfQhth@0_ggossQhw#1t{}
14字一栏：gjgpQt@_gsQw1{fgqfhh0gosh#t}
就4个都提交一下试试，结果全都不对，，，，

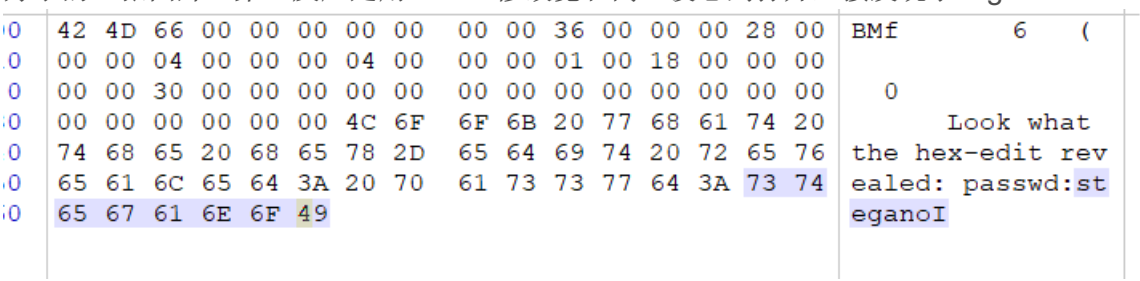
用 ggqht{ggQht_gsQ10jsf#@fopwh} 再试一下 凯撒密码：

看到一个ctf直接提交：

ssctf{ssctf_sec10ver#@rabit}

6，Training-Stegano-1

好小的一张图片，第一反应是用winhex修改宽和高，没想到打开直接发现了flag：



提交：

steganol

7，Excaliflag

用foremost分离没发现东西，用stegsolve看一下：

8，Get-the-key.txt

解压文件 得到一个 没有后缀的 文件 forensic100，看这个名字 我还以为是一个 截取的流量包，后缀改为 pcap 看了一下，没发现什么的，用winhex看一下，猜测是一个压缩包，后缀改为 zip

解压得到 key.txt：

SECCON{@]NL7n+-s75FrET]vU=7Z}

9，glance-50

打开是一个极窄的git图片，那还还用想吗？思路肯定是，先把动态图的每一帧分离出来，再拼接起来

上脚本：

分离：

```
import os
from PIL import Image

def seoaration_gif(gif_file):
    png_dir = gif_file[:-4] + '/'
    os.mkdir(png_dir)
    img = Image.open(gif_file)
    try:
        while True:
            current = img.tell()
            img.save(png_dir+str(current)+'.png')
            img.seek(current+1)
    except:
        pass

if __name__=='__main__':
    gif_file = '123.gif'
    seoaration_gif(gif_file)
```

拼接：

```
#拼接图像.py
from PIL import Image
path = "C:\\Users\\HP\\Desktop\\文档\\Python\\text\\123\\"
save_path = 'C:\\Users\\HP\\Desktop\\文档\\Python\\text\\'

im = Image.new('RGBA',(2*201,600)) #创建新照片

imagefile = [] #存储所有的图像的名称
width = 0
for i in range(0,201):
    imagefile.append(Image.open(path+str(i)+'.png')) #遍历，将图像名称存入imagfile

for image in imagefile:
    im.paste(image,(width,0,2+width,600)) #将图片张贴到另一张图片上
    width = width +2
im.save(save_path+'result.png')
im.show()
```

得到图片：

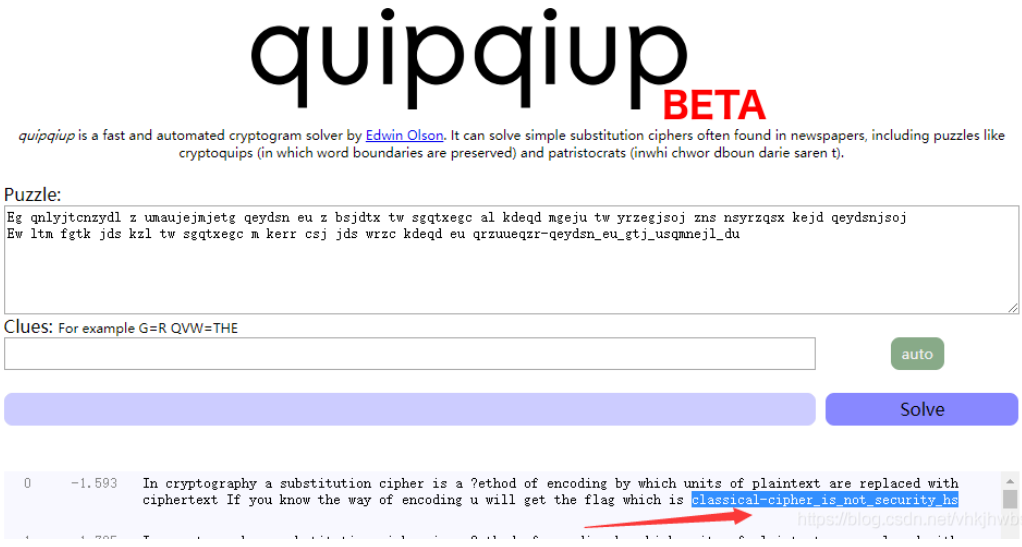
10，4-2

得到一段看起来毫无规律的密文：

```
Eg qnlyjtcnzydl z umaujejmjetg qeydsn eu z bsjdtx tw sgqtxegc al kdeqd mgeju tw yrzegjsoj zns nsyrzqsx kejd
Ew ltm fgtk jds kzl tw sgqtxegc m kerr csj jds wrzc kdeqd eu qrzuueqzr-qeydsn_eu_gtj_usqmnejl_du
```

不知道从何下手，看了一下别人的writeup，发现是 **词频分析**

https://quipqiup.com/



flag{classical-cipher_is_not_security_hs}

11,misc1

打开是一段密文：

```
d4e8e1f4a0f7e1f3a0e6e1f3f4a1a0d4e8e5a0e6ece1e7a0e9f3baa0c4c4c3d4c6fbb9e1e6b3e3b9e4b3b7b7e2b6b1e4b2b6b9e2b1b
```

以为是base64，发现解不出来，再回头仔细看，发现只有 0~9 a~f 十六进制

但是，直接 用 16进制=》ascii 发现是一堆乱码，

常识 ascii 只有 到128 ，而发现这一串16进制每两位化成 十进制 后都大于 128

肯定得减去128后再 转成 ascii：

直接上代码：

```
char = "d4e8e1f4a0f7e1f3a0e6e1f3f4a1a0d4e8e5a0e6ece1e7a0e9f3baa0c4c4c3d4c6fbb9e1e6b3e3b9e4b3b7b7e2b6b1e4b2b
for i in range(0,len(char),2):
    print(chr(int(char[i:i+2],16)-128),end="")
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python37\python3.exe C:/Users/
That was fast! The flag is: DDCTF{9af3c9d377b61d269b11337f330c935f}
Process finished with exit code 0
```

That was fast! The flag is: DDCTF{9af3c9d377b61d269b11337f330c935f}

12，embarrass

下载下来是一个 流量包，在linux中 搜索一下 flag 字符串 看会不会有意外收获：

```
luohao@ubuntu:~/Desktop$ strings './misc_02.pcapng' | grep flag
GET /flag.php HTTP/1.1
GET /flag.doc HTTP/1.1
flag{Good_b0y_W3ll_Done}
flag{Good_b0y_W3ll_Done}
flag{Good_b0y_W3ll_Done}
flag{Good_b0y_W3ll_Done}
      <p>Some antivirus programs mistake XAMPP for a virus, typically flagging the file xampp-manage
r.exe This is a false positive meaning that the antivirus erroneously identified it as a virus, when
 it is not. Before we release each new version of XAMPP we run it through virus scanning software. A
t the moment we are using <a href="http://www.kaspersky.com/virusscanner">Kapersky Online Virus Scan
ner</a>. You can also use the online tool <a href="https://www.virustotal.com/">Virus Total</a> for
scanning XAMPP or send us an email to security (at) apachefriends (dot) org if you find any issue.</
p>
<tr><td class="e">filter.default_flags</td><td class="v"><i>no value</i></td><td class="v"><i>no val
ue</i></td></tr>
<tr><td class="e">windows_tracing_flags </td><td class="v">3 </td></tr>
luohao@ubuntu:~/Desktop$
```

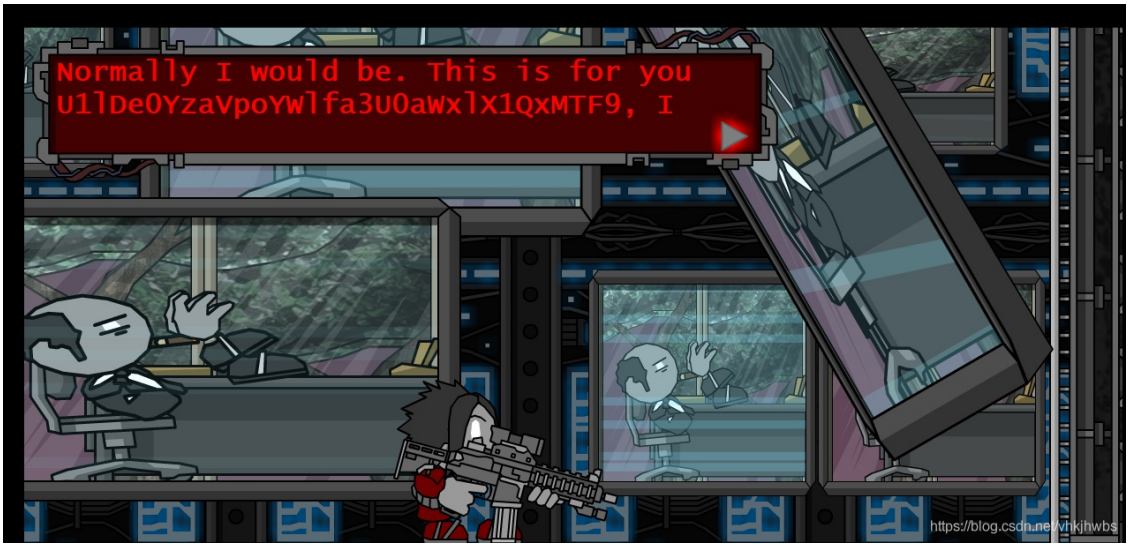

没想到直接出来了

flag{Good_b0y_W3ll_Done}

13， 肥宅快乐题

a.这个题目下载到手是一个.swf后缀的文件
b.用到一个特殊的工具，叫做potplayer
c.把这个.swf文件使用这个工具打开
d.根据提示注意对话就好

点击 右下角的 开始　就会 自动播放　在 放到　57 帧 的时候，　出现了 下面的对话 有一段 base64 ：



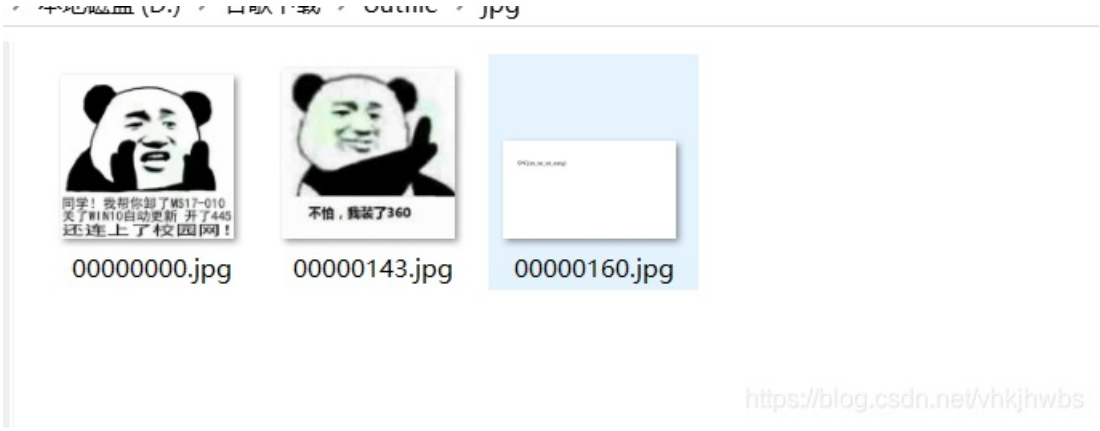
狗日的　不能复

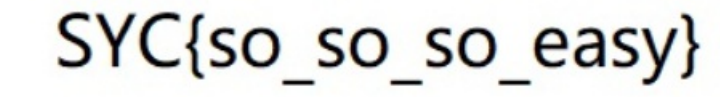
提取出来：U1lDe0YzaVpoYWlfa3U0aWxlX1QxMTF9

base64解码得flag：SYC{F3iZhai_ku4ile_T111}

14，小小的PDF

下载下来是一个 pdf文件 ，不管有没有隐藏文件，先用formost 分离一下，没想到直接 分离出来了三张图片，包含一张flag图片：





至此 我在攻防世界中已经 遇到 三种 pdf 的隐写方式：

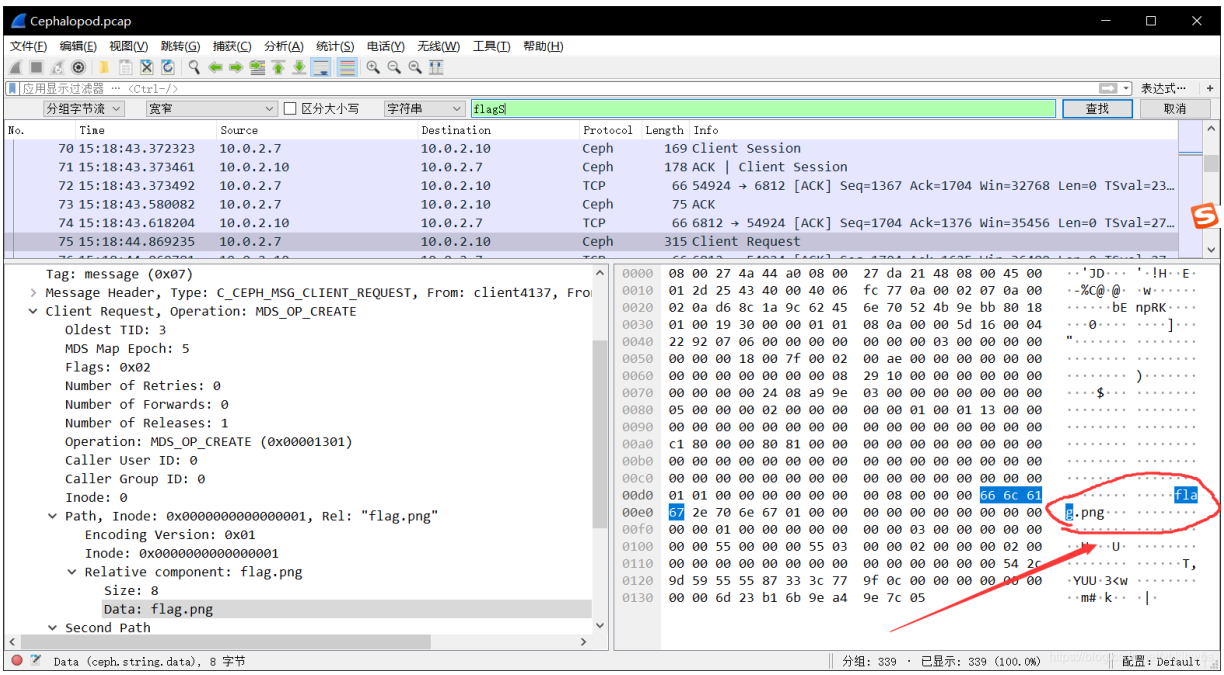
1，新手区 ： pdf ：将flag隐藏在图片的下面　》 需要 先将格式转为 word 再将图片拖开 就能看到 flag

2，新手区 ：stegano ：利用类似水印的方法将flag隐藏在大量文字下面（不清粗具体方法）》 全选复制到 txt文件中就能 显示出密文

3，高手区： 小小的pdf： 嵌入文件 》 直接用 formost 分析 或用 binwalk分离

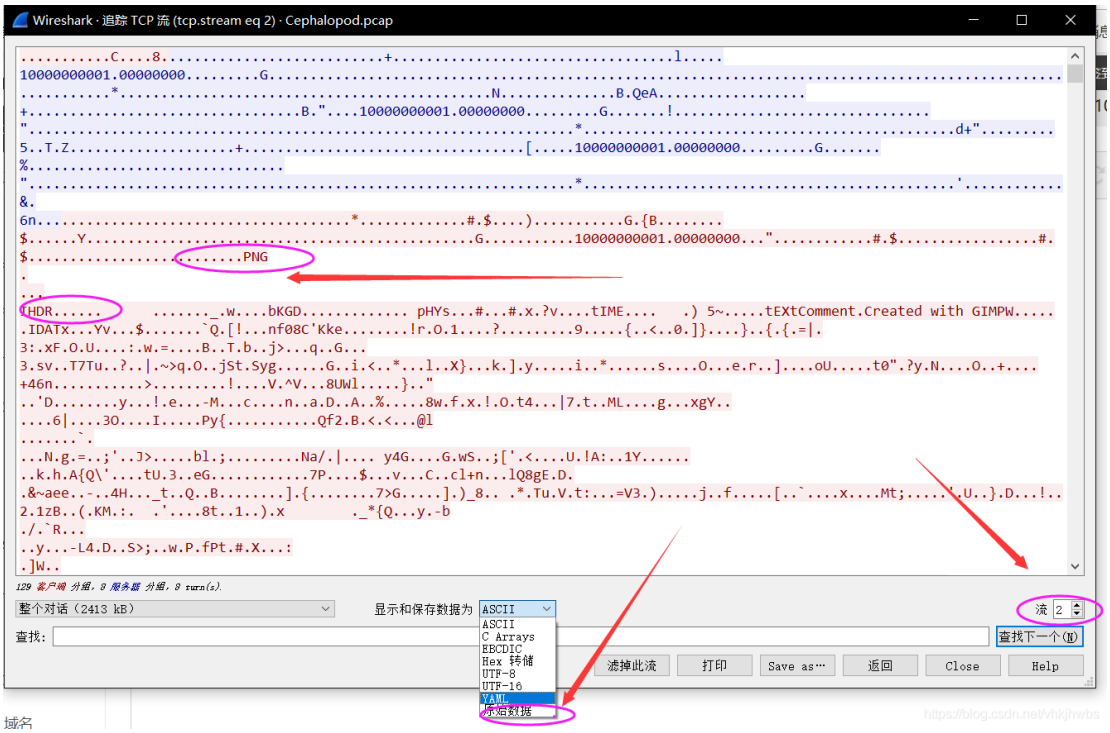
15，Cephalopod

打开数据包，尝试性 在分组字节流中 搜索字符串 flag 然后就看到了有一张 flag.png:

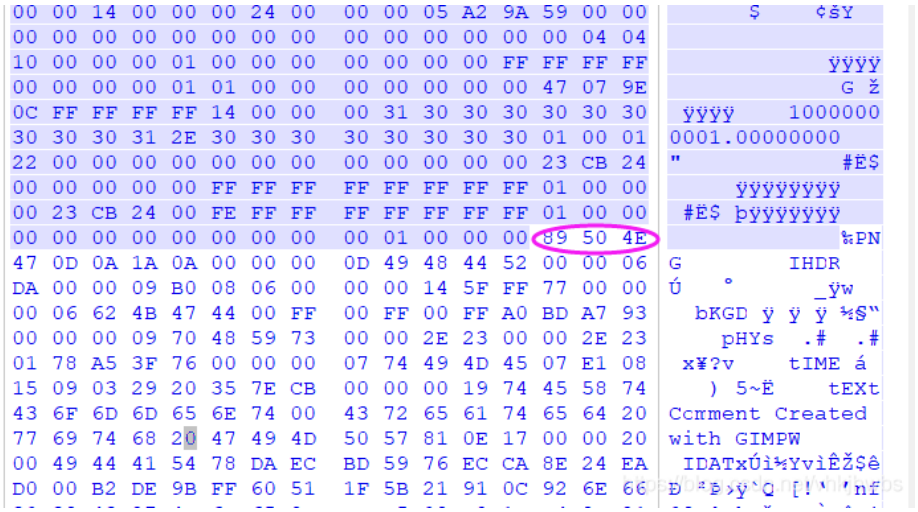


然后 追踪流 （还原原始数据包）发现flag.png 的数据并不在 75 这个包里：

看下一个流：发现了 png 的数据文件：选择 原始数据流 =》save as flag.png （直接保存的话，部分数据会丢失）



然后用 winhex 打开，删除 图片头前面的 无关数据：



然后就看到了flag.png:

HITB{95700d8aefdc1648b90a92f3a8460a2c}

16， hit-the-core

下载下来是一个 core文件：

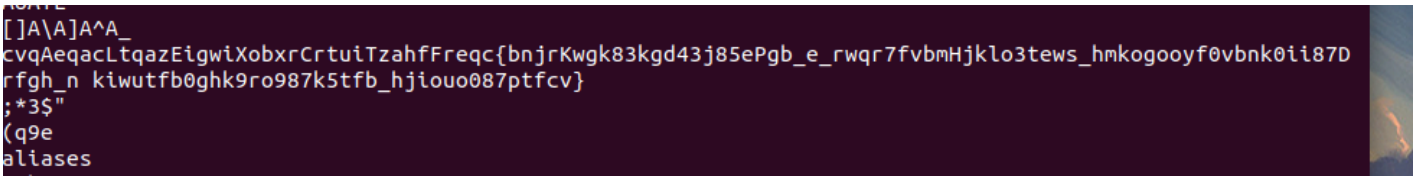
小知识点 core：

core 文件：核心文件（core file），也称**核心转储**（core dump），是操作系统在进程收到某些信号而终止运行时，将此时进程地址空间的内容以及有关进程状态的其他信息写出的一个磁盘文件。这种信息往往用于调试。

核心文件通常在系统收到特定的信号时由操作系统生成。信号可以由程序执行过程中的异常触发，也可以由外部程序发送。动作的结果一般是生成一个某个进程的内存转储的文件，文件包含了此进程当前的运行堆栈信息。有时程序并未经过彻底测试，这使得它在执行的时候一不小心就会遭到破坏。这可能会导致核心转储（core dump）。现在的UNIX系统极少会面临这样的问题。即使遇到，程序员可以通过核心映像调试程序来找到错误原因。

不知如何下手，看别人的博客，发现是用 strings 命令 提取出 文件中的 字符串：

```
strings  123.core
```



竟然能 从这一段 字符串中 发现 几个 大写字母 之间的联系： ALECTF 而且每个字母中间隔了 5个字符，

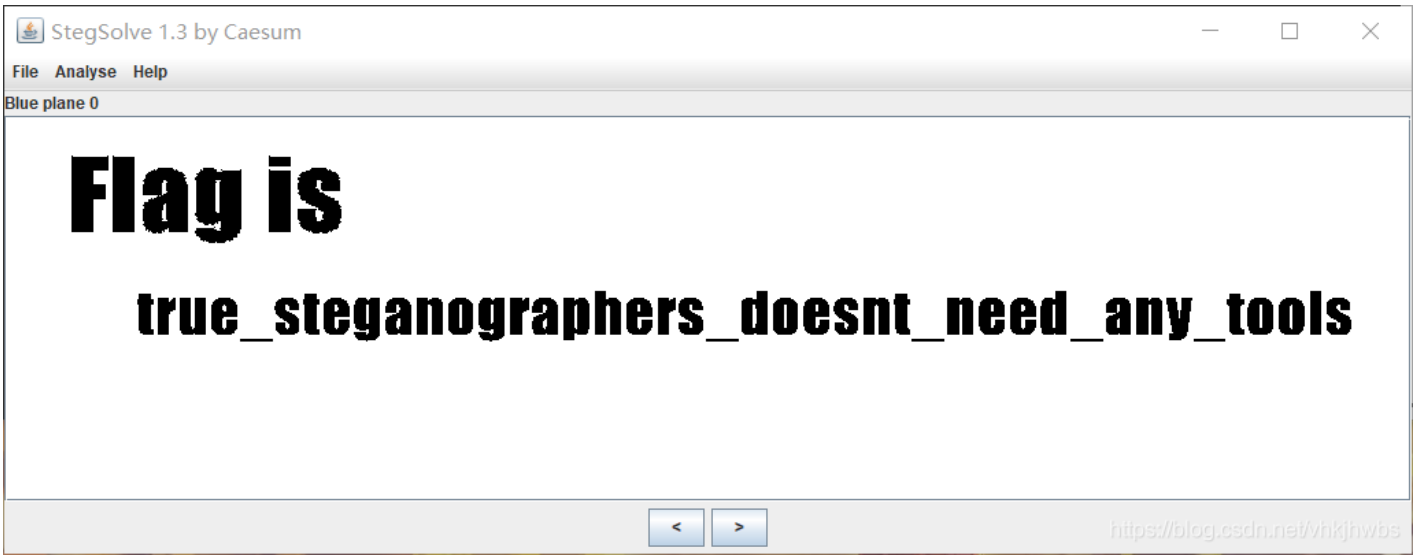
提取这些字符：（真是TM服了，这题都能出）

```
data = 'cvqAeqacLtqazEigwiXobxrCrtuiTzahfFreqc{bnjrKwgk83kgd43j85ePgb_e_rwqr7fvbmHjklo3tews_hmkogooyf0vbnk0
flag = ''
for i in range(3, len(data), 5):
    flag += data[i]

print(flag)
```

得到flag： ALEXCTF{K33P_7H3_g00D_w0rk_up}

17，pure_color

打开是一张白色图片，放进stegsolve 中分析一下：

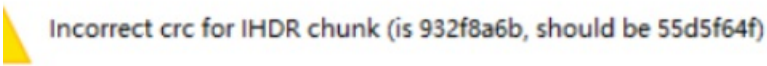


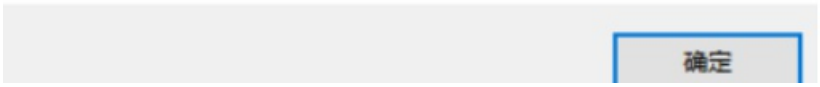
flag{true_steganographers_doesnt_need_any_tools}

18，2-1

是一个png图片 ，打不开 ，文件损坏，用 winhex 打开发现 文件头 是错的 ，把文件头修改为 89 50 4E 47

保存后发现还是打不开，用 tweakpng 打开 有报错：





把文件 IHDR 的crc校验值 修改为 55 d5 f6 4f 后发现 还是打不开，

再仔细看 发现 图片的 宽是 0

根据 crc校验值 算宽是多少：

```
import struct
import binascii
import os

m = open("misc4.png","rb").read()
for i in range(1024):
    c = m[12:16] + struct.pack('>i', i) + m[20:29]
    crc = binascii.crc32(c) & 0xffffffff
    if crc == 0x932f8a6b: #自己根据情况改
        print(i)
```

得到宽度为709，修改宽度为709换算为16进制为 02 c5

flag is wdflag{Png_

C2c_u_kn0W}

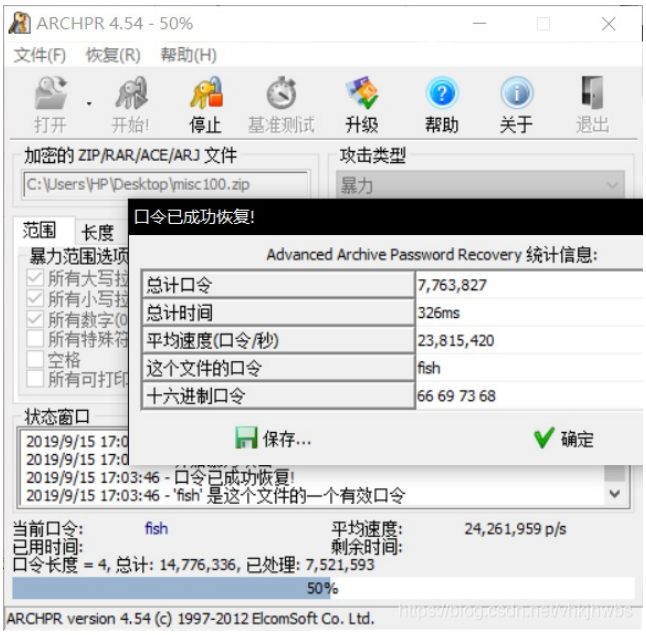
得flag

wdflag{Png_C2c_u_kn0W}

19，János-the-Ripper

解压后得到一个 没有后缀的文件 放进winhex中发现是一个 zip包，后缀改为 zip

解压发现需要密码，直接爆破得到密码 ：fish



解压得到flag：

flag{ev3n::y0u::bru7us?!}

20，2017_Dating_in_Singapore

```
0108152229151617031017243l-050607132027262728-0102030209162330-020916230203100909101724 23-02010814222930-06
```

脑洞是真尼玛大！

12段 数字，每一段都是 有偶数个数字，发现如果两个 字符一组的话，最大的数字的 31，

刚好是日历中最大的数字，然后就在日历上 一个数字一个数字地 连，然后就画出了 下面的字符：

flag：

HITB{CTFFUN}

21, 4-1

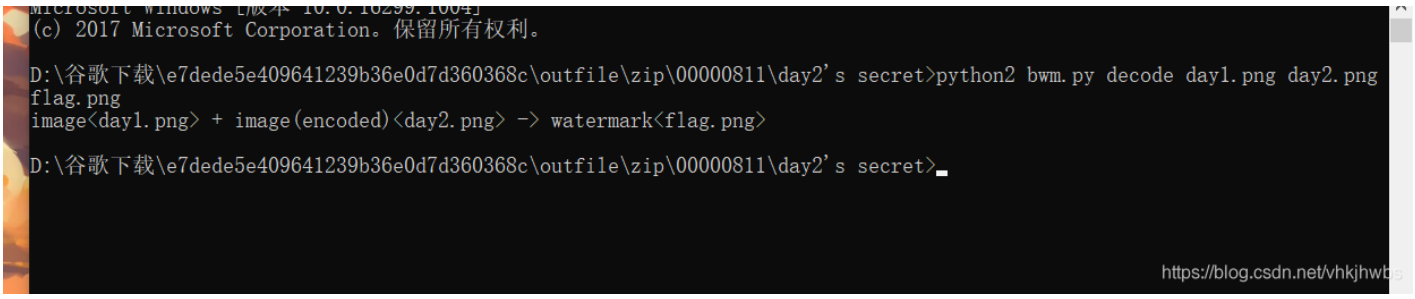
解压得到一张图片，用formost分离出来一个 压缩包，解压得到

得到 两张“一样”的图片，和一个 tips.txt 大意是 指 day2.png 中比 day1.png 多了点“东西”

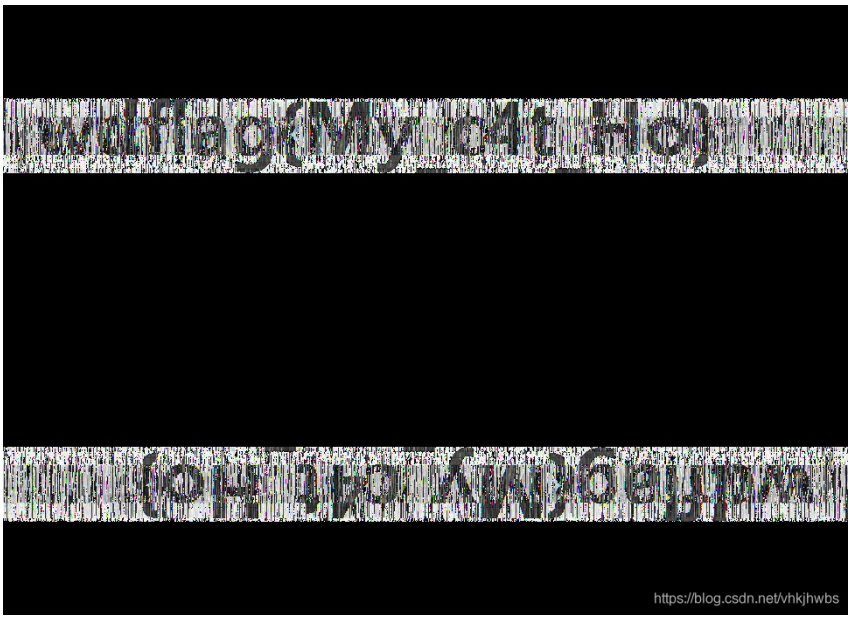
这一看肯定是 水印啊 ，盲水印啊，

直接提取盲水印：

```
python2 bwm.py decode day1.png day2.png flag.png
```



得到水印：



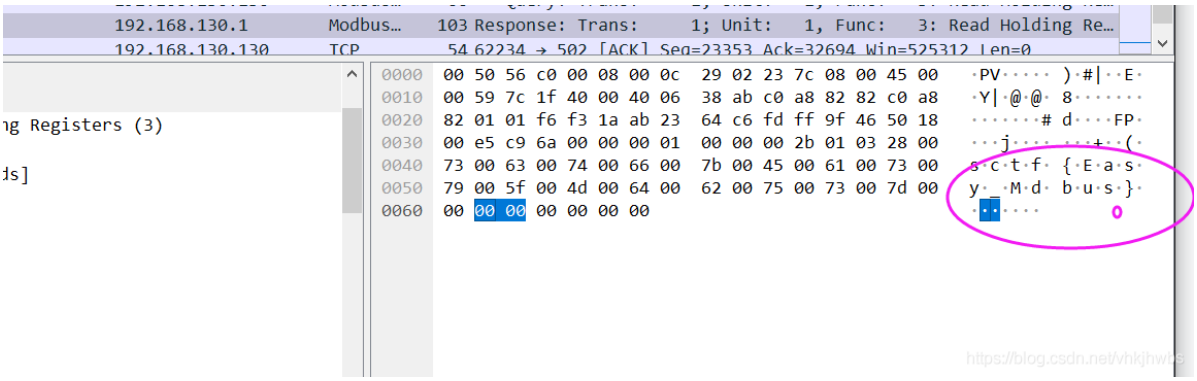
22， 神奇的Modbus

Modbus是一种串行通信协议,

Modbus协议当前存在用于串口、以太网以及其他支持互联网协议的网络的版本。

大多数Modbus设备通信通过串口EIA-485物理层进行

用wireshark 打开流量包，直接在 分组字节流 中 搜索 sctf 字符串 直接出来：



但是 sctf{Easy_Mdbus} 竟然不是 正确答案

正确答案 ：sctf{Easy_Modbus}

23，5-1

是一个 不知道文件类型的 文件 ，用 file 分析一下 显示是类型 为 data ，

看别人的博客，大意是需要用 xortool 工具

先尝试出 key：GoodLuckToYou

```
root@ubuntu:/home/luohao/Desktop# ls
1100  123  misc_02.pcapng  outguess
root@ubuntu:/home/luohao/Desktop# xortool 123
The most probable key lengths:
   2:   12.2%
   5:   11.9%
   9:   9.8%
  13:   22.2%
  20:   6.8%
  22:   6.2%
  26:   12.8%
  30:   4.6%
  39:   7.8%
  52:   5.7%
Key-length can be 3*n
Most possible char is needed to guess the key!
root@ubuntu:/home/luohao/Desktop# xortool 123 -l 13 -c 20
1 possible key(s) of length 13:
Good\tuckToYou
Found 1 plaintexts with 95.0%+ valid characters
See files filename-key.csv, filename-char_used-perc_valid.csv
```


其中-l就是指定密钥长度，-c表示出现频率最高的字符。这个需要根据经验，比如文本内容一般是空格（20），二进制文件一般是00

然后用 脚本进行 解密：

```
import os

c = open("123",'rb').read()
key = "GoodLuckToYou"
def xor(c,k):
  keylen = len(k)
   res = ""
  for pos,c in enumerate(c):
   res +=chr(ord(c) ^ ord(k[pos % keylen]))
  return res
print xor(c,key)
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.16299.1004]
(c) 2017 Microsoft Corporation。保留所有权利。

D:\谷歌下载>python2 w.py
The opening line of the novel famously announces: "It is a truth universally acknowledged, that a single man in possession o
f a good fortune must be in want of a wife." This sets marriage as a central subject鈥nd really, a central problem鈥or t
he novel generally. Readers are poised to question whether or not these single men are, in fact, in want of a wife, or if su
ch desires are dictated by the "neighbourhood" families and their daughters who require a "good fortune". Marriage is a comp
lex social activity that takes political economy, and economy more generally, into account. In the case of Charlotte Lucas,
for example, the seeming success of her marriage lies in the comfortable economy of their household, while the relationship
between Mr and Mrs Bennet serves to illustrate bad marriages based on an initial attraction and surface over substance (econ
omic and psychological). The Bennets' marriage is one such example that the youngest Bennet, Lydia, will come to re-enact wi
th Wickham, and the results are far from felicitous. wdflag{You Are Very Smart}Though the central characters, Elizabeth and
Darcy, begin the novel as hostile acquaintances and unlikely friends, they eventually work to understand each other and them
selves so that they can marry each other on compatible terms personally, even if their "equal" social status remains fraught
. When Elizabeth rejects Darcy's first proposal, the argument of only marrying when one is in love is introduced. Elizabeth
only accepts Darcy's proposal when she is certain she loves him and her feelings are reciprocated. Austen's complex sketchin
g of different marriages ultimately allows readers to question what forms of alliance are desirable, especially when it come
s to privileging economic, sexual, companionate attraction.

D:\谷歌下载>
```


小知识点：

xortool 工具，

xortool.py是基于python的脚本，用于完成一些xor分析，包括:

- 猜想key的长度
- 猜想key的值
- 解密一些经过xoe加密的文件

24，can_has_stdio?

打开文件是一大串brainfuck密文，在线解码得到flag

：http://tool.bugku.com/brainfuck/?wafcloud=2

25，MISCall

不知道是什么文件，用kali看一下，发现是bzip2文件，重命名为 123.bzip2

```
root@kali:~/desktop/share# file 123
123: bzip2 compressed data, block size = 900k
root@kali:~/desktop/share# mv 123 123.bzip2
root@kali:~/desktop/share# ls
123.bzip2  kali.jpg
```

解压：

```
root@kali:~/desktop/share# tar xvf 123.bzip2
ctf/
ctf/flag.txt
ctf/.git/
ctf/.git/description
ctf/.git/refs/
ctf/.git/refs/heads/
ctf/.git/refs/heads/master
ctf/.git/refs/stash
ctf/.git/refs/tags/
```


发现了一个falg.txt文件和 .git文件夹，flag.txt中没有flag

尝试用 git stash 进行恢复以前修改/删除的文件

查看git 记录，给出了一个最近上传的文件，但这个文件并不存在

```
root@kali:~/desktop/share/ctf# git log
commit bea99b953bef6cc2f98ab59b10822bc42afe5abc (HEAD -> master)
Author: Linus Torvalds <torvalds@klaava.Helsinki.Fi>
Date:   Thu Jul 24 21:16:59 2014 +0200

    Initial commit
```

查看修改列表，储存列表中有一条记录

```
root@kali:~/desktop/share/ctf# git stash list
stash@{0}: WIP on master: bea99b9 Initial commit
```

校验一下列表中的存储文件

```
root@kali:~/desktop/share/ctf# git stash show
 flag.txt | 25 +++++++++++++++++++++++++-
 s.py     |  4 ++++
 2 files changed, 28 insertions(+), 1 deletion(-)
```

把上面的文件恢复

直接执行 git stash apply 时，会 提示文件覆盖 自动终止，可以先把flag.txt删除再执行

```
root@kali:~/desktop/share/ctf# git stash apply
error: 您对下列文件的本地修改将被合并操作覆盖：
        flag.txt
请在合并前提交或贮藏您的修改。
正在终止
root@kali:~/desktop/share/ctf# rm -rf flag.txt
root@kali:~/desktop/share/ctf# ls
root@kali:~/desktop/share/ctf# git stash apply
位于分支 master
要提交的变更：
  （使用 "git restore --staged <文件>..." 以取消暂存）
        新文件：    s.py

尚未暂存以备提交的变更：
  （使用 "git add <文件>..." 更新要提交的内容）
  （使用 "git restore <文件>..." 丢弃工作区的改动）
        修改：      flag.txt
```


运行s.py 得到flag：

NCN4dd992213ae6b76f27d7340f0dde1222888df4d3

26，3-1

文件用winhex打开发现是rar文件头，后缀改为rar，解压发现有有个文件名为 ++_++的文件，用记事本打开，发现这是一个流量包，后缀改为 .pcap，用wireshark打开，

在分组字节流 中搜索字符串 flag，

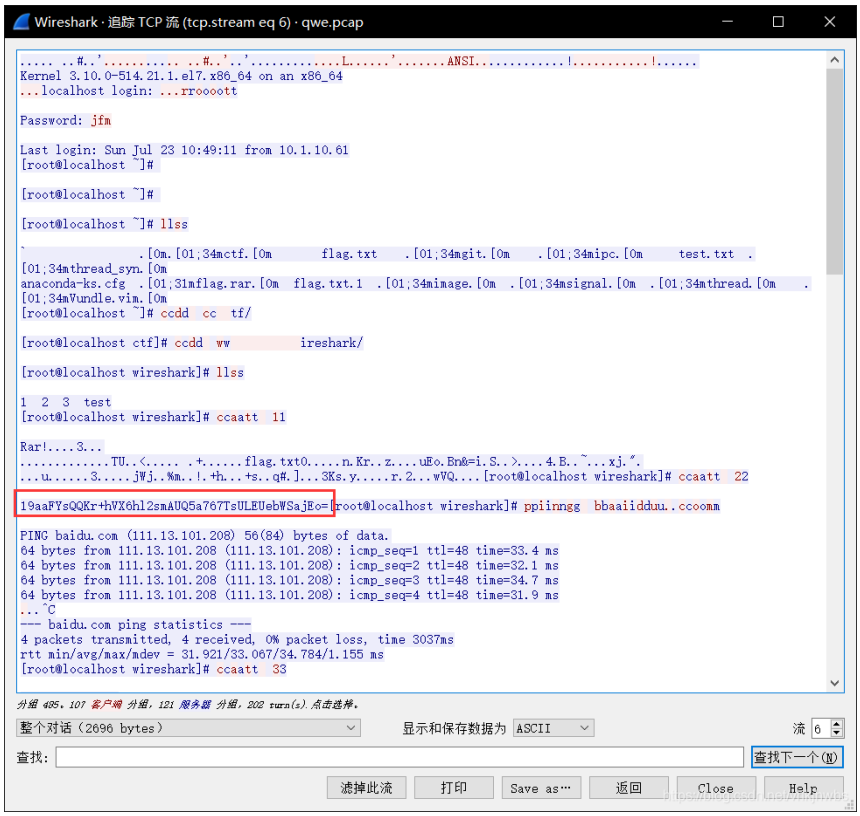
找到了一个 flag.rar文件，

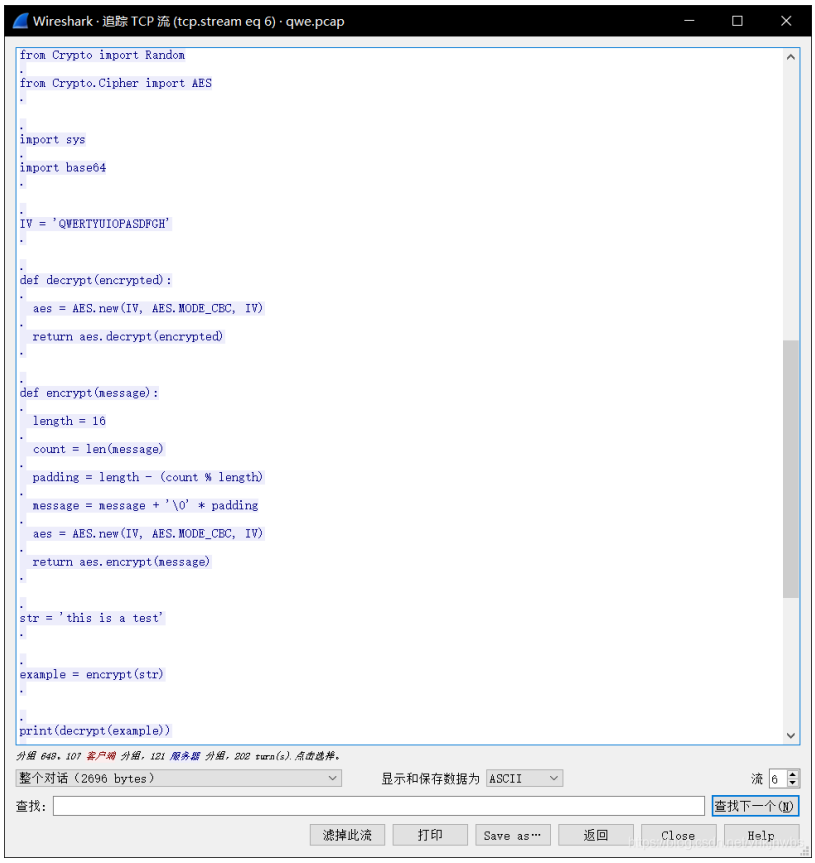
```
GET /flag.rar HTTP/1.1
User-Agent: Wget/1.14 (linux-gnu)
Accept: */*
Host: 10.1.10.61:8000
Connection: Keep-Alive

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.6.0
Date: Wed, 02 Aug 2017 02:53:59 GMT
Content-type: application/octet-stream
Content-Length: 169
Last-Modified: Wed, 02 Aug 2017 02:46:46 GMT

Rar!....3...
.........v...U..<..... ,7RZ....flag.txt0.....-m.l.
..o.-..[V.+..."..............jy.K.;
.....(4....chrQ..I.#k.J...{...*.....{=...'...........'....wVQ....
```


文件 》导出分组字节流 保存为 flag.rar，里面有一个 flag.txt文件

解压发现需要密码，暴力破解无果

会过头来找密码，发现在下一个字节流中（第6个）中，看到了一些管道命令，和一段python加密脚本


Wireshark · 追踪 TCP 流 (tcp.stream eq 6) · qwe.pcap
Kernel 3.10.0-514.21.1.el7.x86_64 on an x86_64
...localhost login: ...rrooootte
Password: jfm
Last login: Sun Jul 23 10:49:11 from 10.1.10.61
[root@localhost ~]#
[root@localhost ~]#
[root@localhost ~]# llss
anaconda-ks.cfg
[root@localhost ~]# ccdd cc tf/
[root@localhost ctf]# ccdd ww ireshark/
[root@localhost wireshark]# llss
1 2 3 test
[root@localhost wireshark]# ccaatt 11
Rar!....3...
19aaFYsQQKr+hVX6hl2smAUQ5a767TsULEUebWSajEo=[root@localhost wireshark]# ppiinngg bbaaiidduu..ccoomm
PING baidu.com (111.13.101.208) 56(84) bytes of data.
64 bytes from 111.13.101.208 (111.13.101.208): icmp_seq=1 ttl=48 time=33.4 ms
64 bytes from 111.13.101.208 (111.13.101.208): icmp_seq=2 ttl=48 time=32.1 ms
64 bytes from 111.13.101.208 (111.13.101.208): icmp_seq=3 ttl=48 time=34.7 ms
64 bytes from 111.13.101.208 (111.13.101.208): icmp_seq=4 ttl=48 time=31.9 ms
--- baidu.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3037ms
rtt min/avg/max/mdev = 31.921/33.067/34.784/1.155 ms
[root@localhost wireshark]# ccaatt 33
整个对话（2696 bytes）
显示和保存数据为 ASCII
流 6
查找:
查找下一个(N)
滤掉此流
打印
Save as…
返回
Close
Help


Wireshark · 追踪 TCP 流 (tcp.stream eq 6) · qwe.pcap
from Crypto import Random
from Crypto.Cipher import AES
import sys
import base64
IV = 'QWERTYUIOPASDFGH'
def decrypt(encrypted):
aes = AES.new(IV, AES.MODE_CBC, IV)
return aes.decrypt(encrypted)
def encrypt(message):
length = 16
count = len(message)
padding = length - (count % length)
message = message + '\0' * padding
aes = AES.new(IV, AES.MODE_CBC, IV)
return aes.encrypt(message)
str = 'this is a test'
example = encrypt(str)
print(decrypt(example))
整个对话（2696 bytes）
显示和保存数据为 ASCII
流 6
查找:
查找下一个(N)
滤掉此流
打印
Save as…
返回
Close
Help

```
from Crypto import Random
from Crypto.Cipher import AES
import sys
import base64

IV = 'QWERTYUIOPASDFGH'

def decrypt(encrypted):
 aes = AES.new(IV, AES.MODE_CBC, IV)
 return aes.decrypt(encrypted)


def encrypt(message):
 length = 16
 count = len(message)
 padding = length - (count % length)
 message = message + '\0' * padding
 aes = AES.new(IV, AES.MODE_CBC, IV)
 return aes.encrypt(message)


str = 'this is a test'
example = encrypt(str)
print(decrypt(example))
```

这是一段加密脚本，密文是19aaFYsQQKr+hVX6hl2smAUQ5a767TsULEUebWSajEo=

写出解密脚本，解密得到：

passwd={No_One_Can_Decrypt_Me}

解密 flag.rar得到 flag：

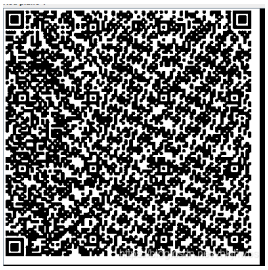
WDCTF{Seclab_CTF_2017}

27，适合作为桌面

是一张图片，二话不说先用foremost分离一下，没分离出来东西

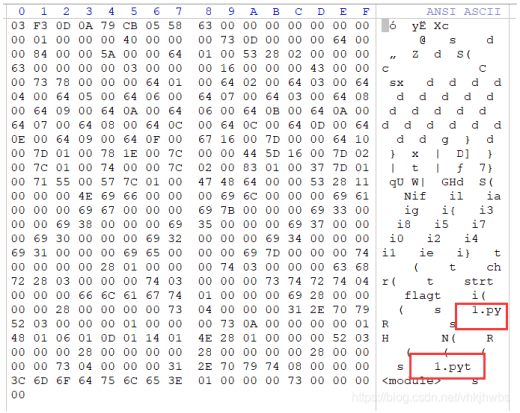
放进stegsolve 里看一下，

发现了一张二维码：



扫描得到一段 16进制：

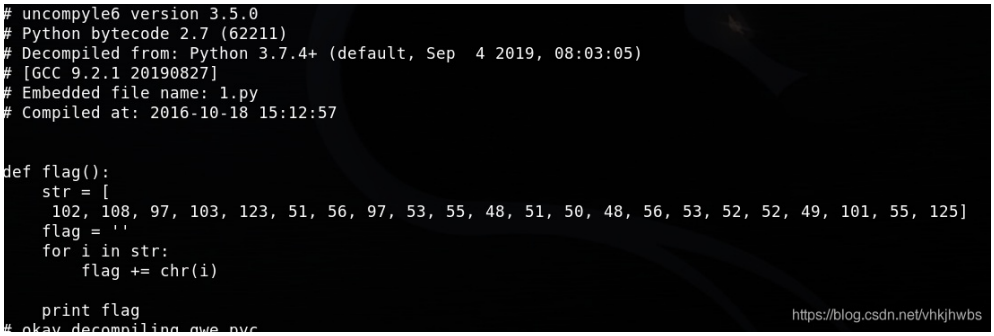
复制到winhex中，发现了 有1.py字符串：



知道这是一个 pyc文件，另存为 1.pyc 放进kali中进行反编译，得到1.py文件：

```
uncompyle6 1.pyc >1.py
```

查看1.py文件：



发现是一个函数，用vim在脚本后面加一个 flag（）调用这个函数，然后执行这个脚本就得到flag：

flag{38a57032085441e7}

28, banmabanma

是一张斑马图片，身上是一段 条形码，用画图拉长后再在线解码

的得到：FLAG IS TENSHINE

flag: flag{TENSHINE}

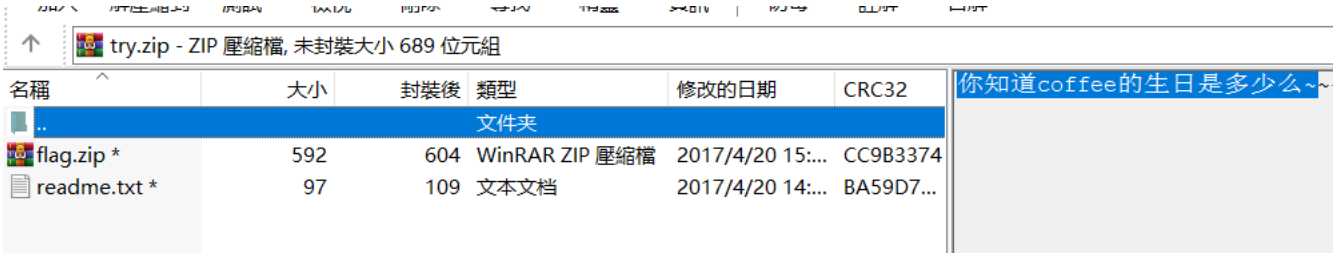
29,我们的秘密是绿色的

这题还是比较难的，需要好多步才能解出来flag

（1）首先得知道有 oursecret这个图片解密软件，然后还得找到解密的密码，

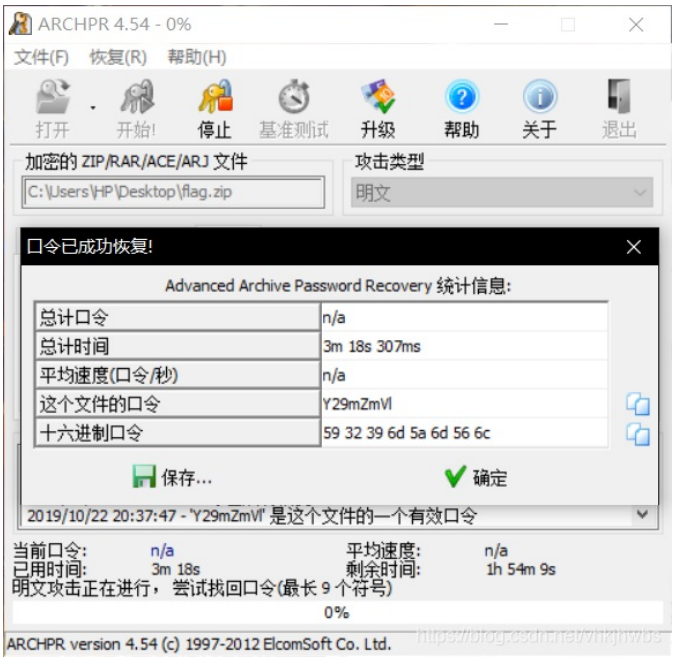
密码经过一番尝试后，发现是 图片中的 绿色的数字 连起来 0405111218192526 ，然后得到一个 try.zip文件

（2）解压try.zip发现需要密码，不过给了点提示：




```
coffee的生日，
别查生日了，直接爆破，纯数字
爆破得到密码：19950822
```

（3）解压后得到 两个文件 flag.zip需要密码，用明文爆破，得到密码：Y29mZmVl



（4）又得到两个 文件，flag.zip 又需要密码

用 ARCHPR爆破时，提示文件没有加密，那应该是 zip伪加密

将 01 改为 00 保存，解压得到flag.txt

```
50 4B 03 04 14 00 00 08  08 00 66 76 94 4A 7D AF  PK        fv”J}¯
72 9F 1E 00 00 00 1E 00  00 00 08 00 00 00 66 6C  rŸ            fl
61 67 2E 74 78 74 2B 4C  49 29 28 2C CF 2B 48 2E  ag.txt+LI)(,Ï+H.
C8 49 CD 53 2D 28 02 B2  E3 AB E3 AB AA B4 52 1C  ÈIÍS-( ²ã«ã«ª´R
D2 0B 6B 01 50 4B 01 02  3F 00 14 00 01 00 08 00  Ò k PK  ?
66 76 94 4A 7D AF 72 9F  1E 00 00 00 1E 00 00 00  fv”J}¯rŸ
08 00 24 00 00 00 00 00  00 00 20 00 00 00 00 00    $
00 00 66 6C 61 67 2E 74  78 74 0A 00 20 00 00 00    flag.txt
00 00 01 00 18 00 E6 FC  D6 7E A2 B9 D2 01 2C E6        æüÖ~¢¹Ò ,æ
57 65 82 B9 D2 01 2C E6  57 65 82 B9 D2 01 50 4B  We,¹Ò ,æWe,¹Ò PK
05 06 00 00 00 00 01 00  01 00 5A 00 00 00 44 00            Z   D
00 00 00 00
```

qddpqwnpcplen%prqwn_{_zz*d@gq}

（4）直接提交，不正确，加flag{}后也不正确，

猜想可能是 栅栏密码 先经过栅栏爆破 得到： qwlr{ddneq_@dpnwzgpc%nzqqpp_*}

（5）再进行 凯撒爆破 得到flag：

flag{ssctf_@seclover%coffee_*}

30，simple_transfer

得到一个流量包，打开，前面的包好像是在进行端口扫描，全是握手包

直接 在分组字节流中搜索 字符串flag ，找到了一个包，追踪流，没发现有用信息

回过头来，再看，往下倒非红色包的部分，发现有 NFS 协议，觉得有点可疑，

过滤看看

发现 在分组列表中 有 file.pdf

```
4297 61.844642  10.0.2.5        10.0.2.4        NFS      210 V4 Call (Reply In 4298) GETATTR FH: 0x0163bd75
4298 61.844943  10.0.2.4        10.0.2.5        NFS      266 V4 Reply (Call In 4297) GETATTR
4300 61.845705  10.0.2.5        10.0.2.4        NFS      210 V4 Call (Reply In 4301) GETATTR FH: 0x0163bd75
4301 61.845888  10.0.2.4        10.0.2.5        NFS      266 V4 Reply (Call In 4300) GETATTR
4303 62.583091  10.0.2.5        10.0.2.4        NFS      230 V4 Call (Reply In 4304) LOOKUP DH: 0x0163bd75/file.pdf
4304 62.583488  10.0.2.4        10.0.2.5        NFS      122 V4 Reply (Call In 4303) LOOKUP Status: NFS4ERR_NOENT
4306 62.583621  10.0.2.5        10.0.2.4        NFS      242 V4 Call (Reply In 4307) SETCLIENTID
4307 62.583814  10.0.2.4        10.0.2.5        NFS      130 V4 Reply (Call In 4306) SETCLIENTID
4308 62.583851  10.0.2.5        10.0.2.4        NFS      174 V4 Call (Reply In 4314) SETCLIENTID_CONFIRM
4314 62.584232  10.0.2.4        10.0.2.5        NFS      114 V4 Reply (Call In 4308) SETCLIENTID_CONFIRM
```


导出分组字节流　保存为 file.pdf

但是我打不开 这个破文件，可能是我导出是姿势有问题，

算了直接 用 foremost 分离 流量包，分离得到 了一个pdf文件：（ma的，下次我直接分解）

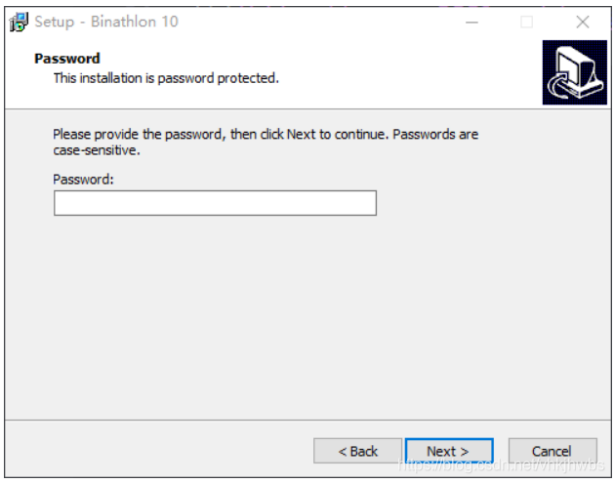
```
HITB{b3d0e380e9c39352c667307d010775ca}
```


HITB{b3d0e380e9c39352c667307d010775ca} 直接提交

31,Just-No-One

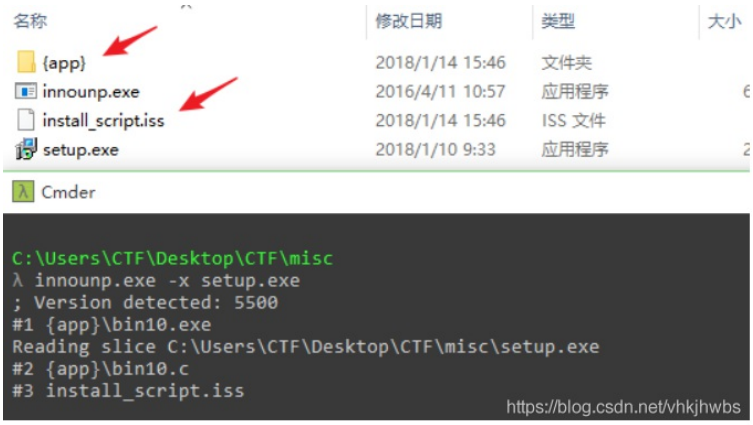
这题我也是服了，这没有点耐心还真找不到

下载得到一个 exe文件，那就安装呗，还能咋地，但发现需要password：



找密码：

> 尝试使用innounp工具解压压缩包。
>
> innounp的官网地址: http://innounp.sourceforge.net
>
> innounp.exe -x setup.exe



在解压出的install_script.iss中发现密码信息

; PasswordHash=888e209e5d76a3135d04b8baccabd7936a0dd376

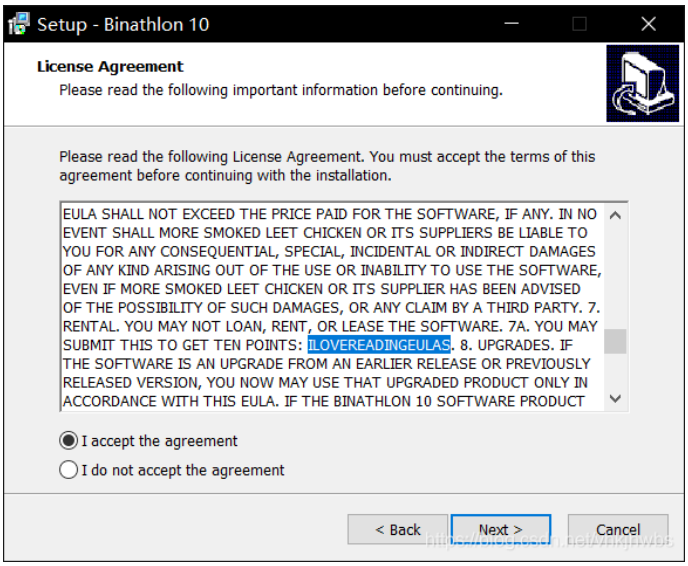
; PasswordSalt=0e559522ecf62077

使用hashcat解密得到密码为:1234567

但实际上并没有什么用，程序安装完打开是这个样子的

提示我们已经看到了flag

最骚的来了，没有任何提示的情况下，能在 安装协议里面找到 正确 flag 的人是真的大佬，反正我做不来



谁能想到 这就是flag： 也没个格式，真想骂这个出题人，

ILOVEREADINGEULAS

32, warmup

考察，明文爆破，盲水印

先将 open_forum.png 压缩为 open_forum.zip，然后明文破解（这里我死活破不开，不知道是怎么回事，我醉了）

然后得到，两张图片，用盲水印提取水印：（脚本自己去github里面找）

```
python bwm.py encode fuli.png fuli2.png res.png
```

flag{bWm_Are_W0nderfu1}

33， Erik-Baleog-and-Olaf

是png头，后缀改为 .png

用 stegsolve 打开 发现在中间的小人的手中，有一个很小的二维码，不断切换找到一张最清的二维码：

用 截图工具截个图：

直接扫，扫不出来，

再把这张图放进 stegsolve中 ，找到一张更清的图片，保存下来

扫描得到flag：flag{#justdiffit}

34，Py-Py-Py

只能说花里胡哨，第一次见到 在 pyc/pyo文件中 隐藏信息的

开始的时候把 pyc文件 放进 kali中 反编译为 py文件

```
uncompyle6 123.pyc > 123.py
```

然后看到一个加密脚本：

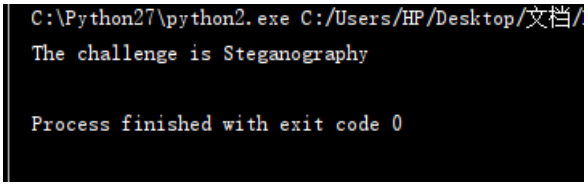
太长了，选出有用的一部分，

```
if __name__ == '__main__':
    while True:
        flag = raw_input('Please input your flag:')
        if flag == crypto(fllag, 'decode'):
            print('Success')
            break
        else:
            continue
```

看到，flag == crypto(fllag,'decode')

直接把这一段 代码删除了 添加一句 print crypto(fllag,'decode')

然后就得大 flag 的值 The challenge is Steganography， 直接提交不对，加flag{}提交仍然不对

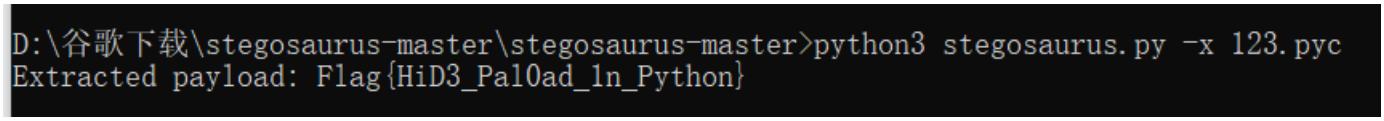


这句话的意思是 挑战是隐写，

去网上搜了一下，还可以在 pyc文件的字节码中 隐藏 信息的隐写方式，

需要用到 stegosaurus 这个工具，github中有，

```
python3 stegosaurus.py -x 123.pyc
```



Flag{HiD3_Pal0ad_1n_Python}

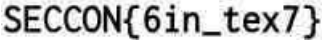
35, reverse_it

放进 winhex中发现是 文件 png 的数据进行了 逆置，（这里的逆置是指，字节的逆置）

```
96 78 C8 00 00 00 41 00  00 00 20 00 23 10 27 00 -xE   A     # '
00 00 91 00 00 00 20 00  13 10 00 00 20 00 10 00   `
00 00 30 00 82 10 A6 00  00 00 10 00 00 00 50 00   0 , ¦       P
B1 10 26 00 00 00 10 00  00 00 50 00 A1 10 00 00 ± &       P ¡
10 00 10 00 00 00 30 00  21 10 70 00 80 00 00 00       0 ! p €
A2 00 D4 D4 00 00 66 96  87 54 2D 00 1E FF 00 00 ¢ ÔÔ  f–‡T-  ÿ
84 00 84 00 10 10 10 00  64 94 64 A4 01 00 0E FF „ „     d”d¤   ÿ
8D FF                                            ÿ
```

数据逆置回来保存为 png， 得到一张逆置的图片：

再把图片逆置过来：



文件逆置：

```
def swap_nibbles(byte):
    return ((byte << 4) | (byte >> 4)) & 0xff

i = open('1234', 'rb')
o = open('1234.txt', 'wb')
o.write(''.join(map(chr, map(swap_nibbles, map(ord, i.read()[::-1])))))
i.close()
o.close()
```

图片翻转：

```
import sys
from PIL import Image

i = Image.open('1234.jpg')
o = Image.new(i.mode, i.size)
idata, odata = i.load(), o.load()

for y in range(i.size[1]):
    for x in range(i.size[0]):
        odata[x, y] = idata[i.size[0] - x - 1, y]

o.save('qwe' + '.jpg')
```

36，mysql

提示：我们在Mysql数据库中存放了flag，但是黑客已经把它删除了。你能找回来flag吗？

考点，用 undrop-for-innodb 恢复 mysql被删的数据

undrop-for-innodb 在github上有，

```
git clone https://github.com/twindb/undrop-for-innodb

cd updrop-for-innodb

make

如果提示
make : bison 命令未找到，则需要安装 bison

apt install -y bison


然后再执行 make 命令 安装
```

（这里我也是第一次见这种题，我尽量详细记录一下）

现在确认一下数据恢复的必要条件：一份ibdata1数据文件，一份要恢复的数据库的表结构

在structure.sql中看到了表的结构：

```
CREATE TABLE `user` (
  `id` smallint(5) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(10) NOT NULL,
  `password` varchar(32) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;
```

在 mysql文件夹中有 数据空间文件ibdata1文件，现在两个需要的文件都有了

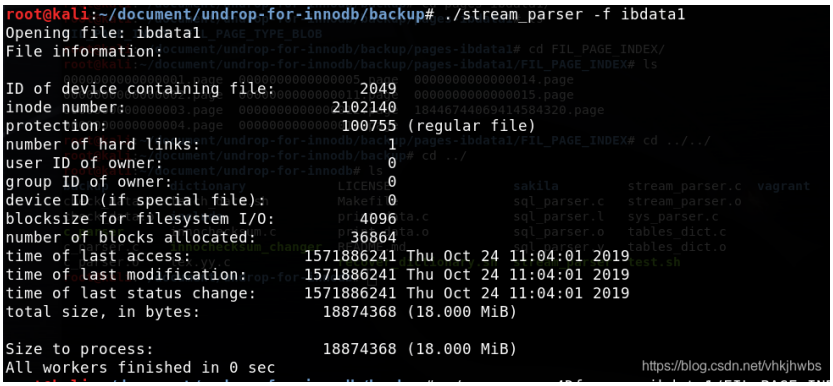
这里我新建了一个 文件夹，backup，然后把需要用到的几个文件复制进去

c_parser ， dictionary ， stream_parser ，structure.sql，ibdata1

（1）解析数据文件

首先，由于mysql将Innodb驱动的数据使用B+tree索引在了数据空间文件ibdata1中，所以需要使用stream_parser工具进行解析：

```
./stream_parser -f ibdata1
```



解析完成后，可以看到同目录下生成一个pages-ibdata1目录，其中包含两个子目录，一个是包含按索引排序的数据页目录，另一个是包含相关类型的数据目录：

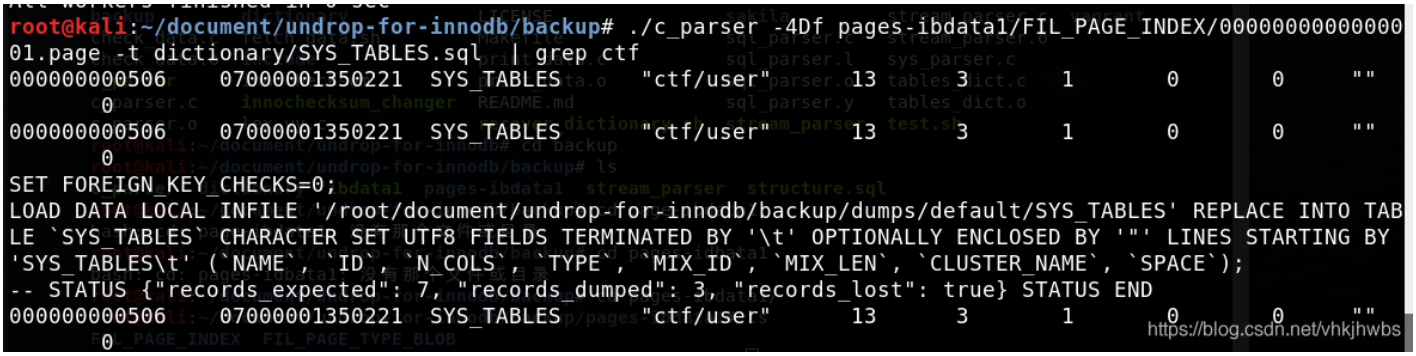
我们下面将主要关注的是第一个子目录即索引好的数据页目录，因为我们要恢复的数据就在里面，其中第一个页文件（0000000000000001.page）里包含所有数据库的表信息和相关的表索引信息，类似一个数据字典，可以使用项目提供的一个脚本recover_dictionary.sh将其内容放到一个test数据库里详细的查看，这里就不做演示了。

（2）解析页文件

既然第一个页文件包含所有数据库表的索引信息，我们就需要先解析它，以模拟mysql查询数据的过程，最终才能找到要恢复的数据。c_parser工具可以用来解析页文件，不过需要提供该页文件的一个内部结构（表结构）。

项目根目录下有个dictionary目录，里面就包含数据字典用到相关表结构，如用来解析第一个页文件的表结构在SYS_TABLES.sql文件

```
./c_parser -4Df pages-ibdata1/FIL_PAGE_INDEX/0000000000000001.page -t dictionary/SYS_TABLES.sql  | grep ctf
```

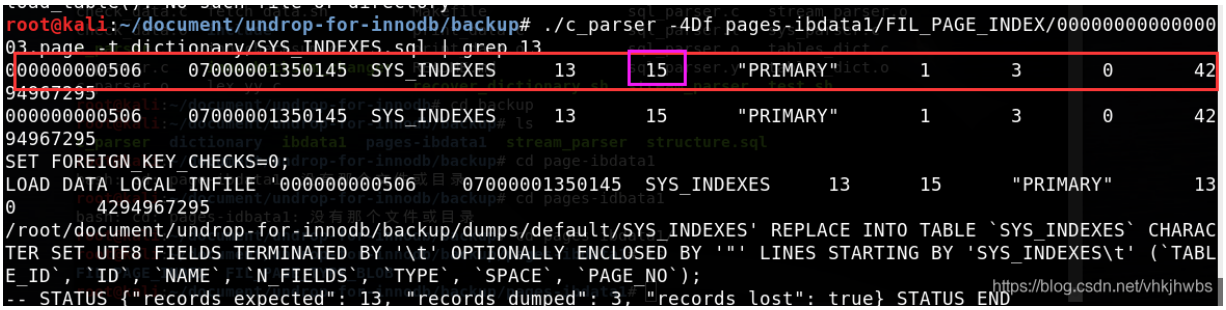


该命令使用c_parser工具解析数据库表索引信息并过滤出我们想要恢复的有关ctf的文件

我们看到 user这个表的索引值为 13，通过这个索引值，再到另外一张表去查询该user表所有的索引信息

该表的结构在"dictionary/SYS_INDEXES.sql"文件中可以看到，而此表对应的数据页文件是第三个数据页0000000000000003.page

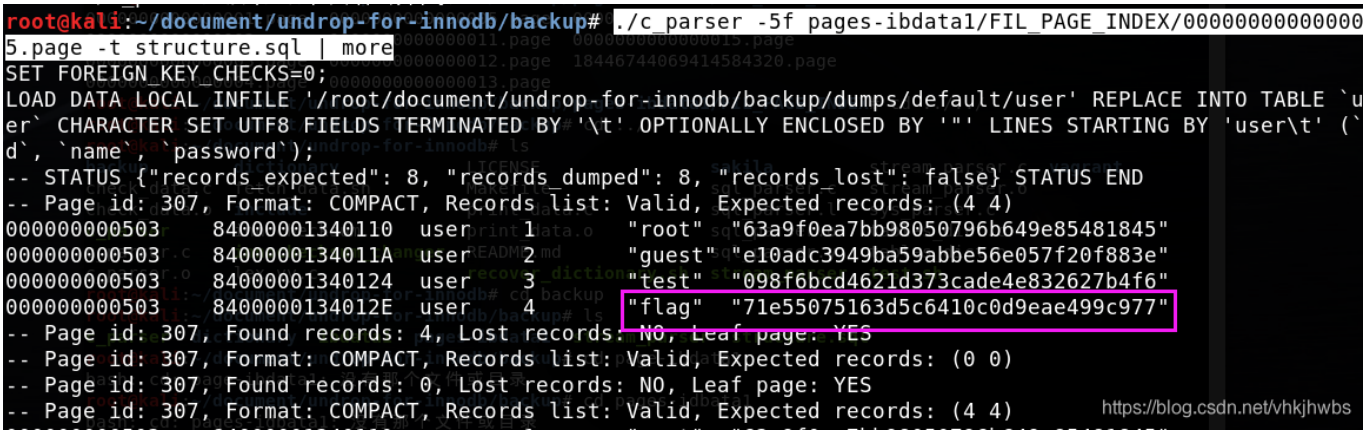
```
./c_parser -4Df pages-ibdata1/FIL_PAGE_INDEX/0000000000000003.page -t dictionary/SYS_INDEXES.sql | grep 13
```



这里找到一条user 的索引信息，其在mysql 存储中的索引值为 15，此索引值编号对应的数据页文件中，即存储了该索引的全部数据

此处我们选择的是主键索引对应的数据页文件进行解析（另外一个索引键应该也可以，只不过方法可能需要有所区别），终于顺利解析见到了激动人心的数据：

```
./c_parser -5f pages-ibdata1/FIL_PAGE_INDEX/0000000000000015.page -t structure.sql | more
```



得到flag：

71e55075163d5c6410c0d9eae499c977

写在最后：（上面的很多题，我也是第一次遇到，也是边刷边学习）

我也是一个刚学习的web安全的小白，也是通过刷题写writeup来提升自己，写的不好的地方勿喷，转身离开就好，如果觉得有帮助到你，就点个赞就好