Have Fun misc

主讲人：Xu

Version: 0.2.5

引言

不是脑洞的CTF题---有趣的MISC

我期望中的Misc 不是脑洞的题 至少要让人知道要做什么吧

如何做好Misc题呢?
细心：50%

套路：40%

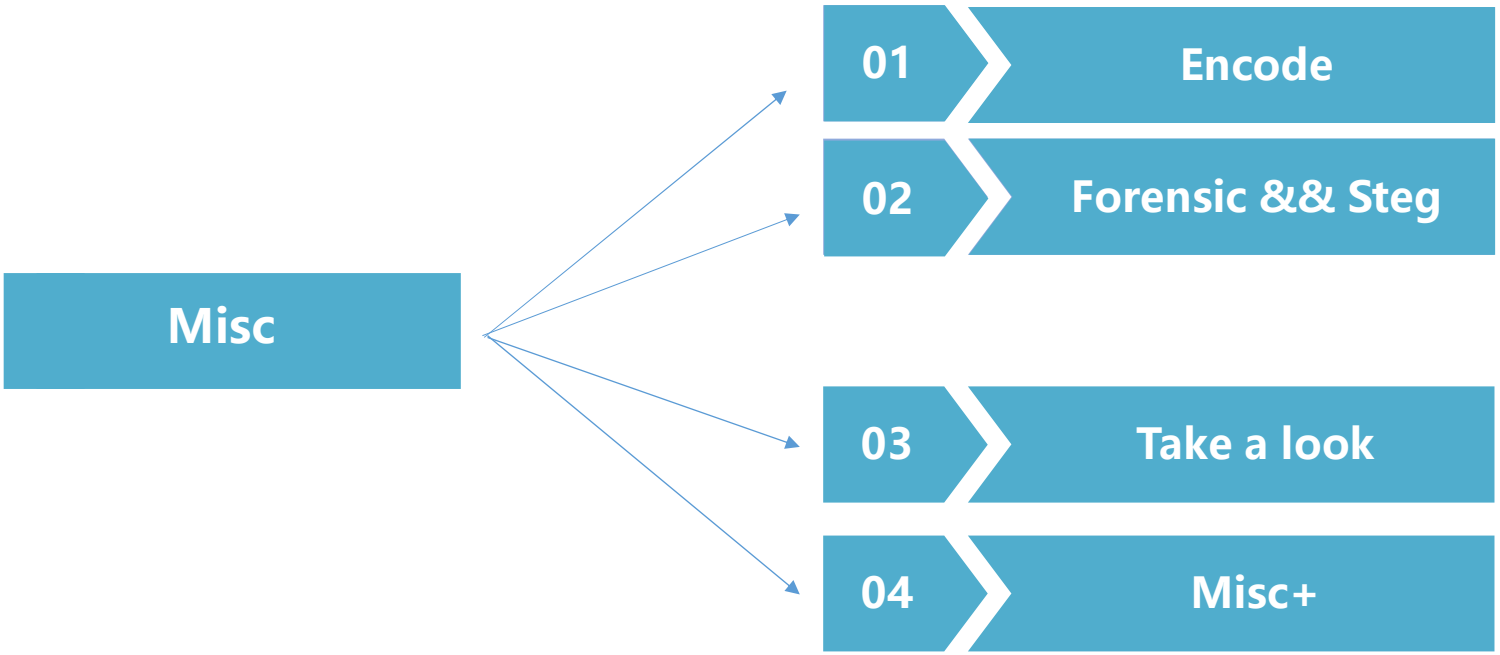
散发的思维：10%
讲真的脑洞的MISC题在我眼里都是浪费时间

0x00: 赛题总览

XCTF

Recnet Misc Challenge

分类



0x00:Recon

信息搜集

- 侠义Recon
  - 社工
- 广义Recon
  - AnyWhere

Misc	Points	Solved by	First solvers
Get help	10	53%	dcua Kesatria Garuda The Cat is #1!!
Give feedback	15	36%	Epic Leet Team w0pr LEGOFAN
4w1h	100	19%	gn00bz STT BreakPoint
Recon 1	140	14%	dephault Steampunkers Avidya
Recon 2	190	5%	PPP Snatch The Root CS-WAT
ctfclicker	220	1%	PPP Samurai 217

0x00:Recon

信息搜集

Recon_1:

Someone has attacked your site. We have attached a log collected from the time of the attack. This task is split into two parts. The end goal (Recon 2) is to find the full name of the attacker. The flag for Recon 1 is not the name of the attacker. Recon 1 flag will appear as 9447{...} on your screen when you find it.
Hint! The attacker has gone on vacation, and won't be reading or responding to emails. You do not need to email anyone for this challenge.

Recon_2:

Find the attackers full name. See attached file on Recon 1.

0x01:Recon

社工

我是一个有贴吧的男人！！！

社工库 ⟶ 贴吧 ⟶ QQ ⟶ 猜域名

于是查看留言板内容，发现提示域名的注册人联系号码就是Flag。然后上万网查询比赛官网WHOIS得到答案。

目录 content

0x01:Encode

编码 解码以及相互间的转换：

列举的常见编码：

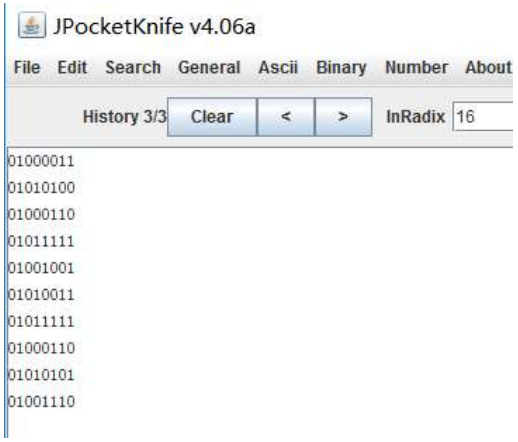
- Bin(二进制)
- Dec(十进制)
- Hex（十六进制）
  - 条形码
  - 二维码
- Base全家桶：
  - Base64 --> Base编码图片
  - Base32
  - Base16
- URL 编码：
- ASCII编码
- 摩尔斯编码
- 曼彻斯特编码/差分曼彻斯特

0x01:Encode

二进制编码的Misc：

二进制编码是用预先规定的方法将文字、数字或其他对象编成二进制的数码，或将信息、数据转换成规定的二进制电脉冲信号。

二进制编码与字符串之间的转换



0x01:Encode

细谈二进制：

0x01:Encode

二进制编码的Misc：

二进制编码与十进制之间的转换

```
>>> bin(100)
'0b1100100'
>>> int("1100100",2)
100
>>>
```

十进制编码与十六进制之间的转换

```
>>> hex(100)
'0x64'
>>> int("0x64",16)
100
>>>
```

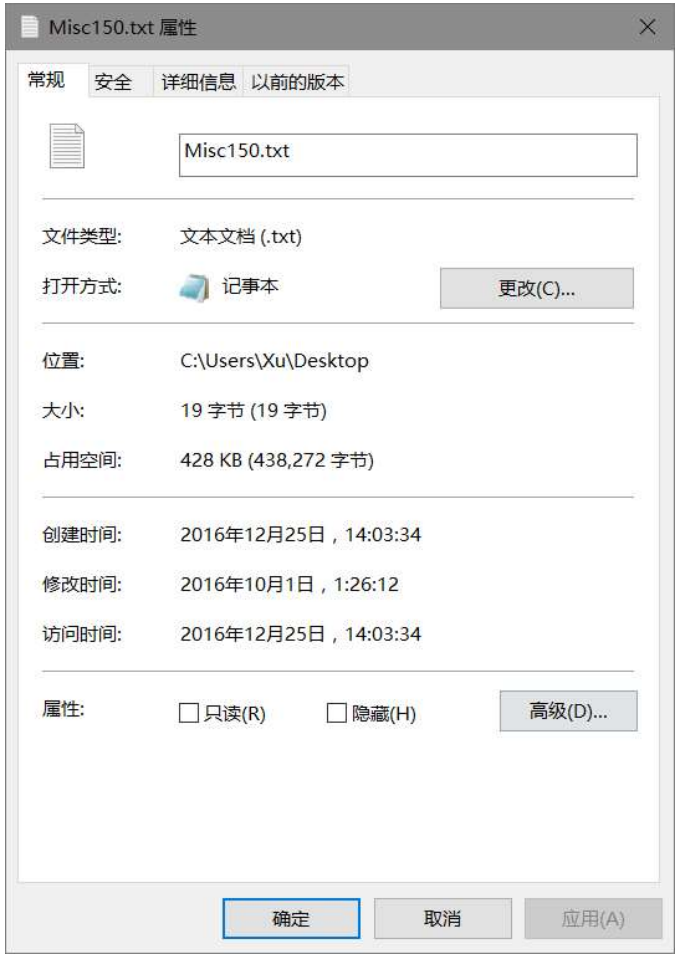
0x01:Encode

二进制转化：

L-CTF:神奇的压缩文件-150

文件名：Misc150_aa0060814bbf.rar

NTFS数据流隐写+二进制的转化

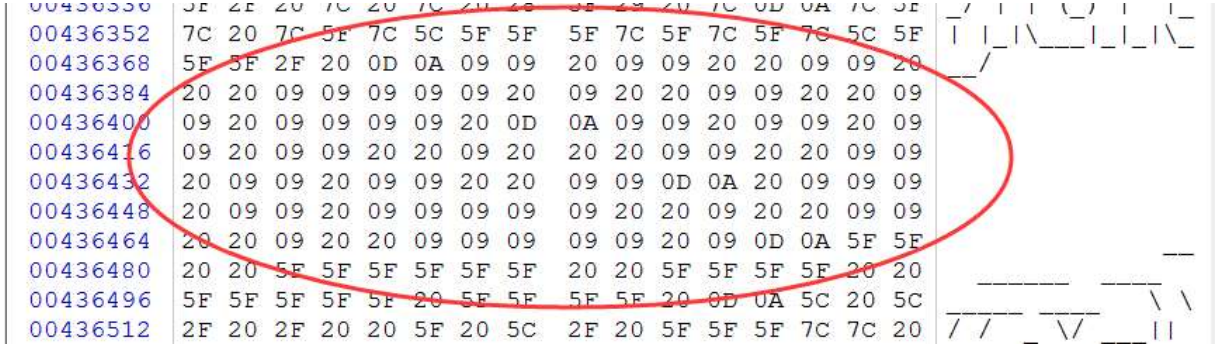


0x01:Encode

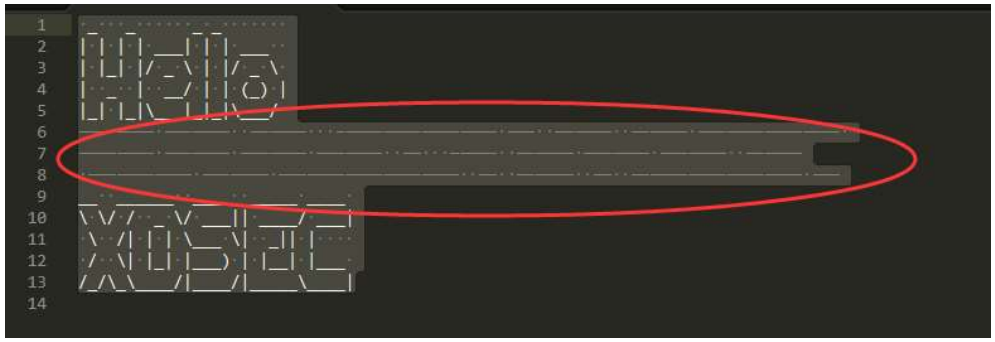
二进制转化：

L-CTF:神奇的压缩文件-150

文件名：Misc150_aa0060814bbf.rar



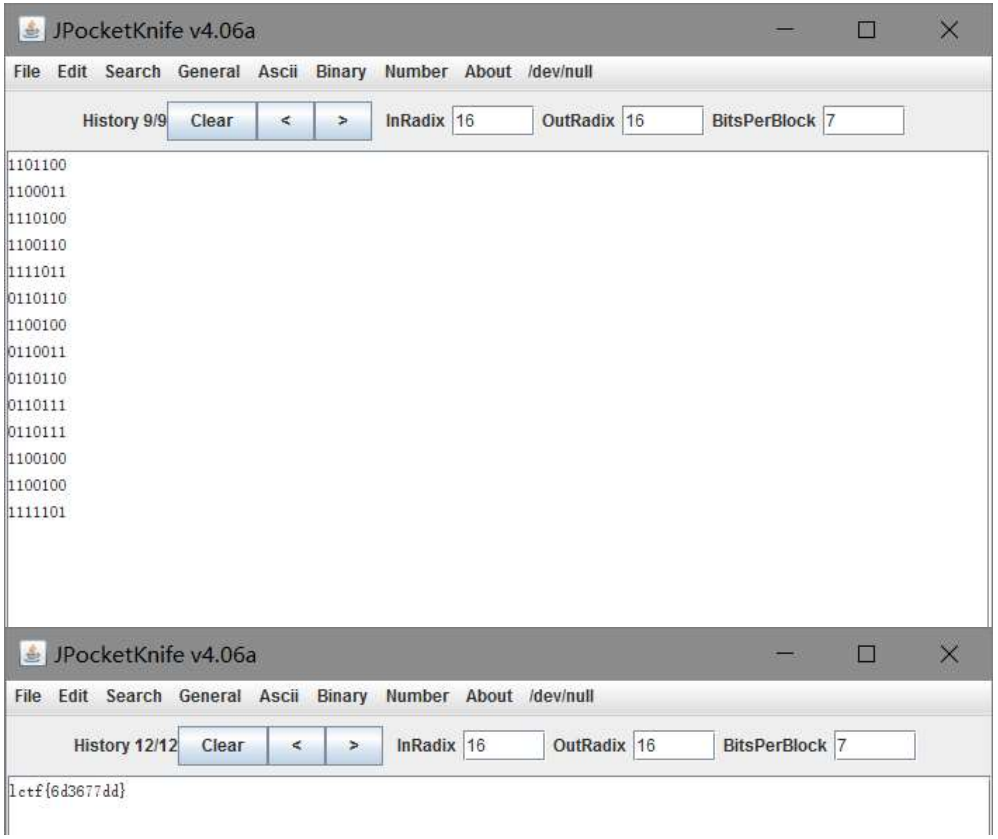
```
00436352  7C 20 7C 5F 7C 5C 5F 5F  5F 7C 5F 7C 5F 7C 5C 5F  | |_|\___|_|_|\_
00436368  5F 5F 2F 20 0D 0A 09 09  20 09 09 20 20 09 09 20  __/
00436384  20 20 09 09 09 09 09 20  09 20 20 09 09 20 20 09
00436400  09 20 09 09 09 09 20 0D  0A 09 09 20 09 09 20 09
00436416  09 20 09 09 20 20 09 20  20 20 09 09 20 20 09 09
00436432  20 09 09 20 09 09 20 20  09 09 0D 0A 20 09 09 09
00436448  20 09 09 20 09 09 09 09  09 20 20 09 20 20 09 09
00436464  20 20 09 20 20 09 09 09  09 09 20 09 0D 0A 5F 5F                __
00436480  20 20 5F 5F 5F 5F 5F 5F  20 20 5F 5F 5F 5F 20 20    ______  ____
00436496  5F 5F 5F 5F 5F 20 5F 5F  5F 5F 20 0D 0A 5C 20 5C  _____ ____    \ \
00436512  2F 20 2F 20 20 5F 20 5C  2F 20 5F 5F 5F 7C 7C 20  / /  _ \/ ___||
```



0x01:Encode

二进制转化：

七个一组转化为



0x01:Encode

二进制转化 ---> 二维码：

```
11111110001000011011111111000001011100101101000001101110101000000000101110110111010010000000010111011011101
01110110100101110110000010101011011010000011111111010101010101111111000000001011101110000000011010011000001
01001110110111101010100100001110000000000010100000000100100110100010011100111101110011111000011101111100011
01010001100111000010101000110100011110101100000101000101100000110111011001000011100111001000010111111101000
00000110101001000111101111111011100001101011011100000100001100110001111010111010001101001111100001011101011
00011101001110010111010010011101101100011000001011000110100011000111111101101011011101 1011
```

共计625位 ---> 25 * 25

0x01:Encode

二进制转化：

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
01100000110011111100111111111000111111000001110000111111111011000001101111111100111111111000001110000110000110
01100000110110000110110000011011000011000110110000000110000110000011011000001101100000110001101100001100110
01100000110110000000110000011011000000001100011000001100001100000110110000011011000001100110001100001111000
01111111110011111100111111111000111111001100000110000110000110000011011111111001100000110110000011000011000
01100000110000000110110000011000000011011111111100001100001100000110110001100011000001101111111110000110000
01100000110110000110110000011011000011011000001100001100001100000110110000110011000001101100000110000110000
01100000110011111100111111111000111111001100000110000110000011111111001100000110111111110011000001100001100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

共计1177位

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
01100000110011111100111111111000111111000001110000111111111011000001101111111100111111111000001110000110000110
01100000110110000110110000011011000011000110110000000110000110000011011000001101100000110001101100001100110
01100000110110000000110000011011000000001100011000001100001100000110110000011011000001100110001100001111000
01111111110011111100111111111000111111001100000110000110000110000011011111111001100000110110000011000011000
01100000110000000110110000011000000011011111111100001100001100000110110001100011000001101111111110000110000
01100000110110000110110000011011000011011000001100001100001100000110110000110011000001101100000110000110000
01100000110011111100111111111000111111001100000110000110000011111111001100000110111111110011000001100001100
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
...
```

0x01:Encode

Base64

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.

0x01:Encode

Base64

Base64 是怎么编码的

- 字符对应ASCII转换成八位二进制
- base64的基础单位是 3*8bit的二进制，若是不够3*8bit则在后面添加0字节（padding）直至满足
- 3*8bit的二进制转换成4*6bit的二进制
- 4*6bit的二进制转换成十进制
- 对照base64表把十进制转换成字符

0x01:Encode

Base64

Base64 是怎么解码的

- 检查base64编码后面有几个等于号
- 把字符串按照base64表转换成4*6的倍数
- 删除等于号的个数*8的bit
- 按照6个bit一组转成字符

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

0x01:Encode

Base64

Base64 隐写

By reason of his fallen divinity

QnkgcmVhc29uIG9mIGhpcyBmYWxsZW4gZGl2aW5pdHm=
QnkgcmVhc29uIG9mIGhpcyBmYWxsZW4gZGl2aW5pdHk=

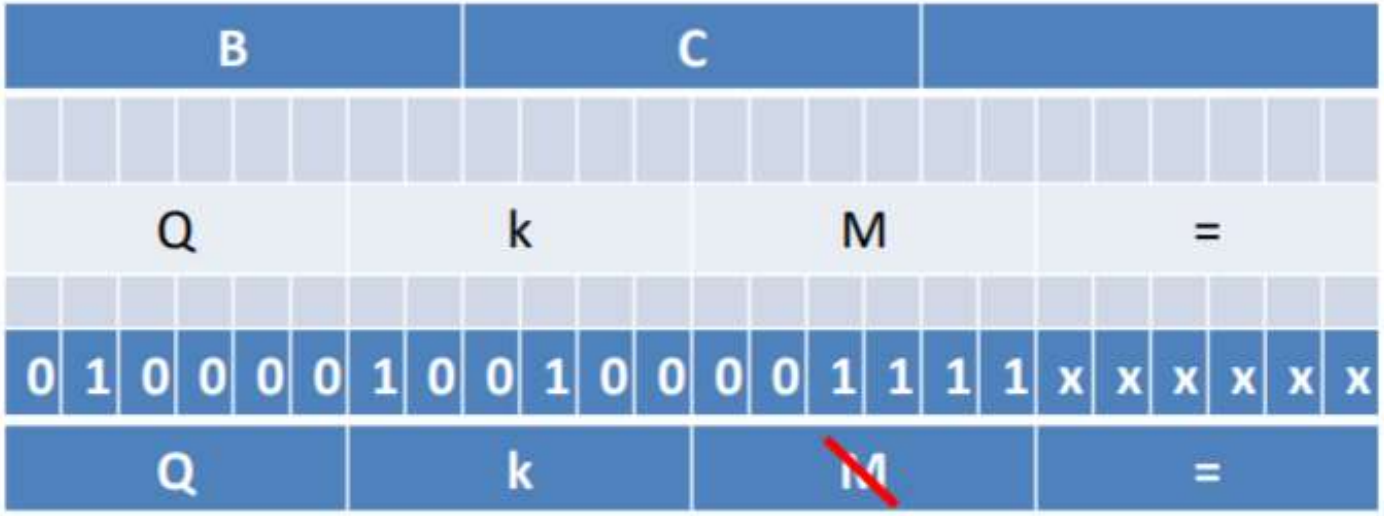
删除等于号的个数8的bit

- 2014 Olympic base64
- 2017 NJCTF ???

0x01:Encode

Base64

Base64 隐写



0x01:Encode

Base64

Base64 隐写

```
1
2  # -*- coding: utf-8 -*-
3  b64chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
4  with open('text.txt', 'rb') as f:
5      bin_str = ''
6      for line in f.readlines():
7          stegb64 = ''.join(line.split())
8          rowb64 =  ''.join(stegb64.decode('base64').encode('base64').split())
9          offset = abs(b64chars.index(stegb64.replace('=','')[-1])-b64chars.index(rowb64.replace('=','')[-1]))
10         equalnum = stegb64.count('=') #no equalnum no offset
11         if equalnum:
12             bin_str += bin(offset)[2:].zfill(equalnum * 2)
13         print ''.join([chr(int(bin_str[i:i + 8], 2)) for i in xrange(0, len(bin_str), 8)]) #8位一组
14
```

0x01:Encode

Morse

音频中的Morse

0x01:Encode

图形码：

https://online-barcode-reader.inliteresearch.com/

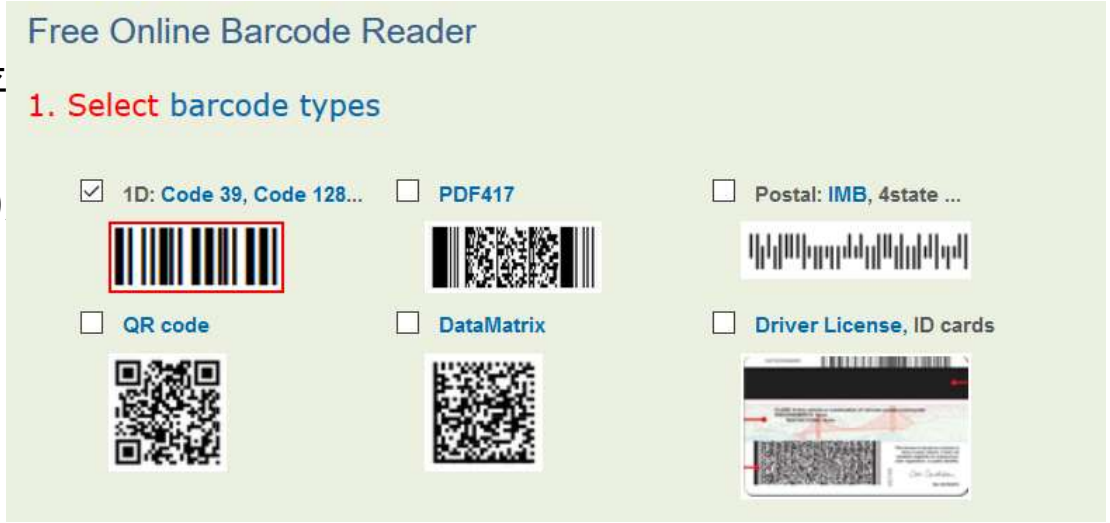
1.条形码：宽度不等的多个黑条和空白，按照一定的编 码规则排列，用以表达一组信息的图形标识符

国际标准 – EAN-13: 商品条码标准，13位数字 – Code-39: 39字符 – Code-128: 128字符

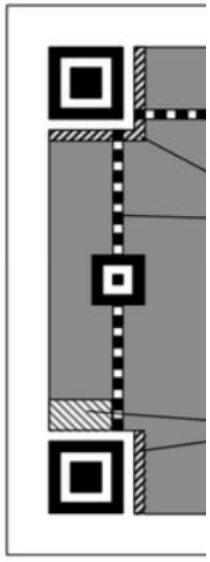
2.二维码：用某种特定几何图形按一定规律在 平息

堆叠式/行排式二维条码：Code 16K、Code 49

一般CTF赛题中图形码基本包含在这其中



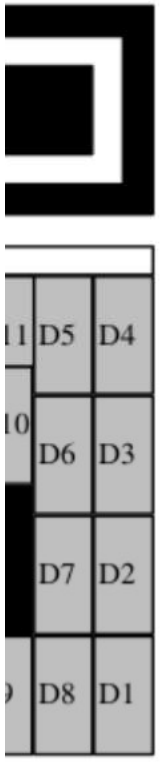
0x01:Encode



QR Coc
Format

e

6



0x01:Encode

CTF中的编码：

https://www.hackfun.org/CTF/coding-and-encryption-of-those-brain-holes-in-CTF.html

0x01:Encode

CTF中的编码

Brainfuck

+++++ +++++ [->++ +++++ +++<] >++++ .---. +++++ ++..+ ++.<+ +++++ +++++ [->++ +++++

Jsfuck

[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]

Jother

+!![])())[!![]+!![[]+]+!![]]+(!![]+[])[+!![]]+({}[[]]+[])[!![] +!![])())[!![]+!![[]+]+!![]]+(!![]+[])[+!![]]+({}[[]]+[])[!![]

0x01:Encode

CTF中的编码

1.asp混淆加密

2.php混淆加密

3.css/js混淆加密

4.VBScript.Encode混淆加密

0x01:Encode

CTF中的编码

编码类密码

- baaba aabbb aabaa aabab abbab babab
- XF AD DA AF XD XG GA FG XA FX DX GX DG FA DX FF

0x01:Encode

工具小结

http://tool.ph0en1x.com/hashtool/tools.html#conv/

JPK

Shell

Python

目录 content

0x02:Forensic&Steg

常见取证对象：

1.PCAP流量包分析：
- 普通的网络流量(最为常见)
- 蓝牙数据包
- USB数据包(鼠标，键盘数据包)
- ……
- XNUCA的投影仪数据包……

2.各种图片文件：JPG PNG
3.音频，视频：MP3,WAV,AVI
4.压缩包：RAR，ZIP，7z
5.磁盘文件：img
6.内存镜像
7.PDF,WORD…

综述：

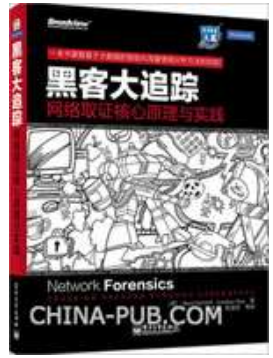
目的一般为发现文件中包含的隐藏字符串(代表需要取证的机密信息)

通常夹杂着文件修复

而这些搜寻的字符串常常又与隐写加密结合在一起

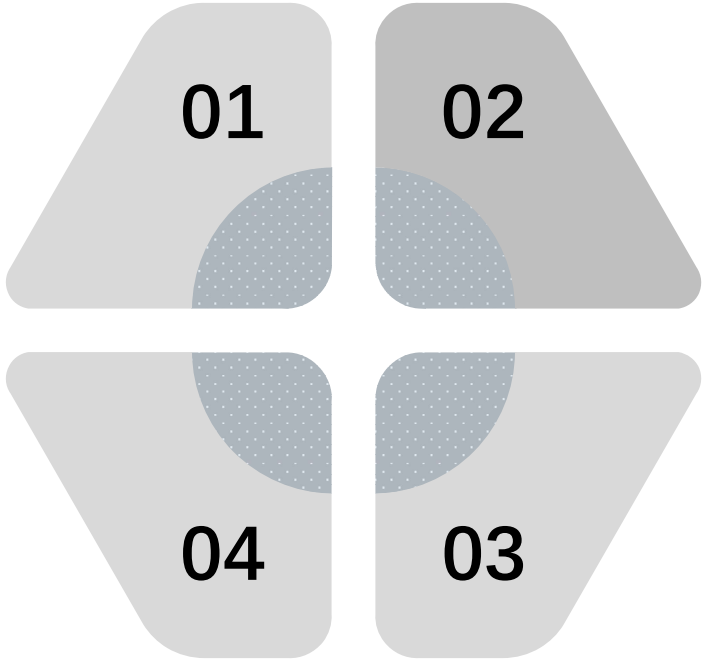
对文件中的特殊编码要有一定的敏感度

对文件16进制的熟悉



0x02:Forensic&Steg

前置技能



Encode

Python

File Format

Tools

0x02:Forensic&Steg

前置技能

Encode

- Base64
- Hex
- Bin

0x02:Forensic&Steg

前置技能

Python

- 字符串处理
- 二进制数据处理
- 文件处理
  - ZIP
  - PNG
  - PCAP
- 网络编程

0x02:Forensic&Steg

前置技能

File Format

Hex signature	ISO 8859-1	Offset	File extension	Description
a1 b2 c3 d4 d4 c3 b2 a1	....	0	pcap	Libpcap File Format[1]
0a 0d 0d 0a	....	0	pcapng	PCAP Next Generation Dump File Format[2]
ed ab ee db	....	0	rpm	RedHat Package Manager (RPM) package [3]

https://en.wikipedia.org/wiki/List_of_file_signatures

0x02:Forensic&Steg

前置技能

Tools

- File：　　用来鉴定文件类型
  - Windows　trid.exe
- Strings：　　查看文件中可见字符串，一般用来找到hint
- Binwalk,foremost：　　用于分析文件，自动切割文件
- Winhex，010Editor：　　16进制文件编辑器
- Grep,awk:　　关键信息检索提取

0x02:Forensic&Steg

工具详解

Strings

Display printable strings in [file(s)]

0x02:Forensic&Steg

工具详解

Binwalk

Binwalk is a fast, easy to use tool for analyzing, reverse engineering, and extracting firmware images

根据文件头识别包含文件
并进行自动化提取

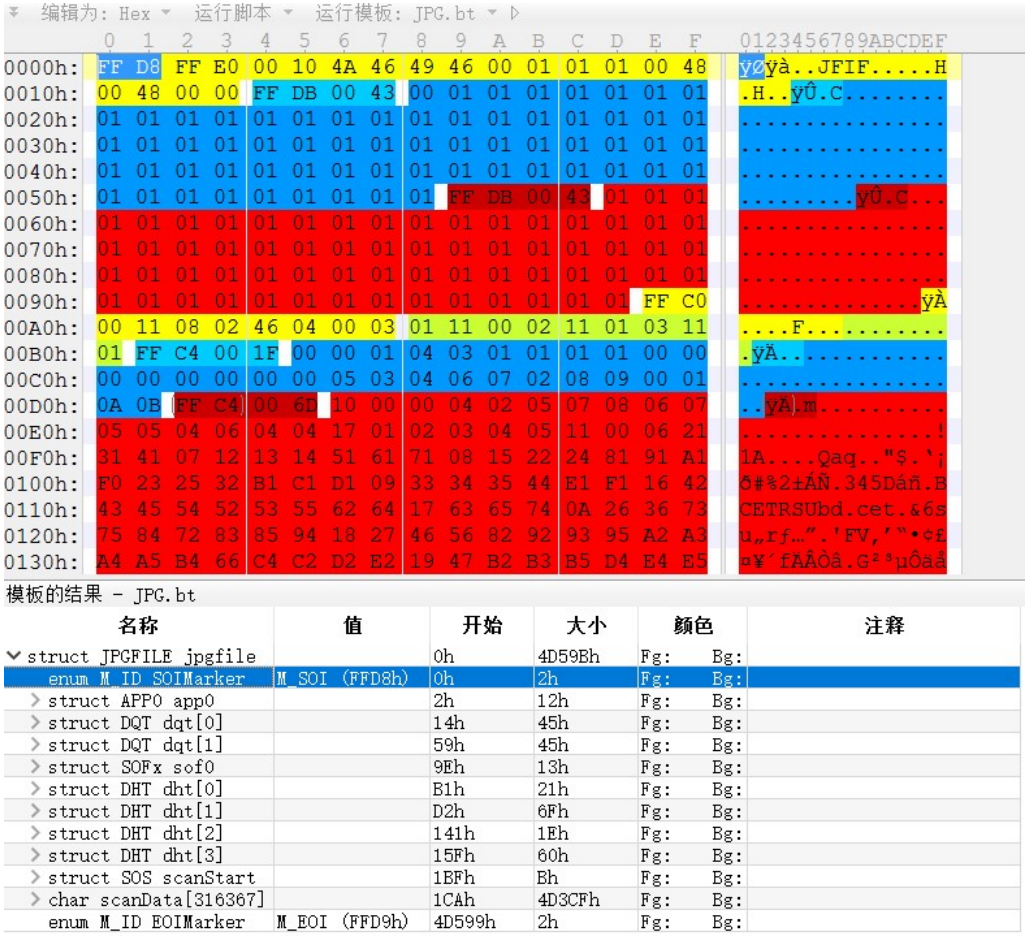
搜索嵌入的二进制镜像文件的代码及文件

0x02:Forensic&Steg

工具详解

010 Editor

分析文件16进制
强大的模板功能



0x02:Forensic&Steg

Picture

MetaData

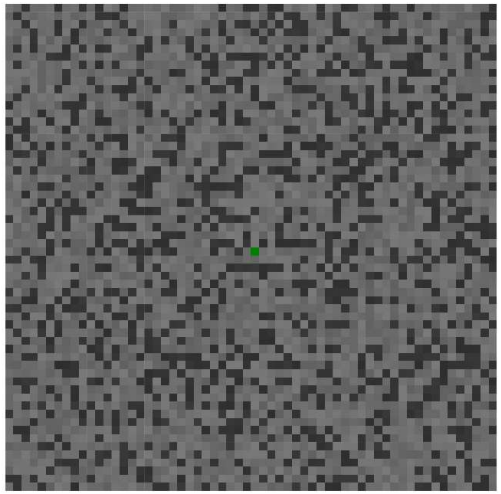
- Identifiy 获取图形文件的格式与特性
  - -format print various properties and other settings associated with an image
  - https://www.imagemagick.org/script/escape.php
- Strings
- ExifTools

0x02:Forensic&Steg

Picture

图像 ⟷ 像素

```python
from PIL import Image
import re

x = 307 #x坐标  通过对txt里的行数进行整数分解
y = 311 #y坐标  x*y = 行数

rgb1 = [****]
print len(rgb1)/3
m=0
for i in xrange(0,x):
    for j in xrange(0,y):

        line = rgb1[(3*m):(3*(m+1))]#获取一行
        m+=1
        rgb = line

        im.putpixel((i,j),(int(rgb[0]),int(rgb[1]),int(rgb[2])))#rgb转化为像素
im.show()
im.save("flag.png")
```



0x02:Forensic&Steg

PNG文件格式

PNG文件署名域+标准数据块(+辅助数据块):

PNG文件署名：**89 50 4e 47 0d 0a 1a 0a**

名称	字节数	含义
Length	4字节	数据域长度
Chunk Type Code	4字节	A-Za-z组成
Chunk Data	可变	数据
CRC	4字节	循环冗余码

0x02:Forensic&Steg

PNG文件格式

标准数据块

文件头数据块IHDR：	图像数据的基本信息	png数据流中第一个出现 仅有一个
调色板数据块PLTE：	索引彩色图像有关	需要在IDAT块之前
标准数据块：	图像数据块IDAT	实际的图像数据，可多个连续
图像结束数据：	IEND	文件结束 00 00 00 00| 49 45 4E 44 | AE 42 60 82

IHDR在最前，IEND在最后，其余数据块随意放置

0x02:Forensic&Steg

PNG文件格式

IHDR

域的名称	字节数	说明
Width	4 bytes	图像宽度，以像素为单位
Height	4 bytes	图像高度，以像素为单位
Bit depth	1 byte	图像深度： 索引彩色图像：1，2，4或8 灰度图像：1，2，4，8或16 真彩色图像：8或16
ColorType	1 byte	颜色类型： 0：灰度图像, 1，2，4，8或16 2：真彩色图像，8或16 3：索引彩色图像，1，2，4或8 4：带α通道数据的灰度图像，8或16 6：带α通道数据的真彩色图像，8或16
Compression method	1 byte	压缩方法(LZ77派生算法)
Filter method	1 byte	滤波器方法
Interlace method	1 byte	隔行扫描方法： 0：非隔行扫描 1： Adam7(由Adam M. Costello开发的7遍隔行扫描方法)

0x02:Forensic&Steg

PNG文件格式

辅助数据块:

PNG文件格式规范制定的10个辅助数据块是:
(1) 背景颜色数据块bKGD(background color)。
(2) 基色和白色度数据块cHRM(primary chromaticities and white point)。所谓白色度是指当R = G = B = 最大值时在显示器上产生的白色度。
(3) 图像γ数据块gAMA(image gamma)。
(4) 图像直方图数据块hIST(image histogram)。
(5) 物理像素尺寸数据块pHYs(physical pixel dimensions)。
(6) 样本有效位数据块sBIT(significant bits)。
(7) 文本信息数据块tEXt(textual data)。
(8) 图像最后修改时间数据块tIME (image last-modification time)。
(9) 图像透明数据块tRNS (transparency)。
(10) 压缩文本数据块zTXt (compressed textual data)

0x02:Forensic&Steg

PNG文件格式

PNG文件格式中的数据块

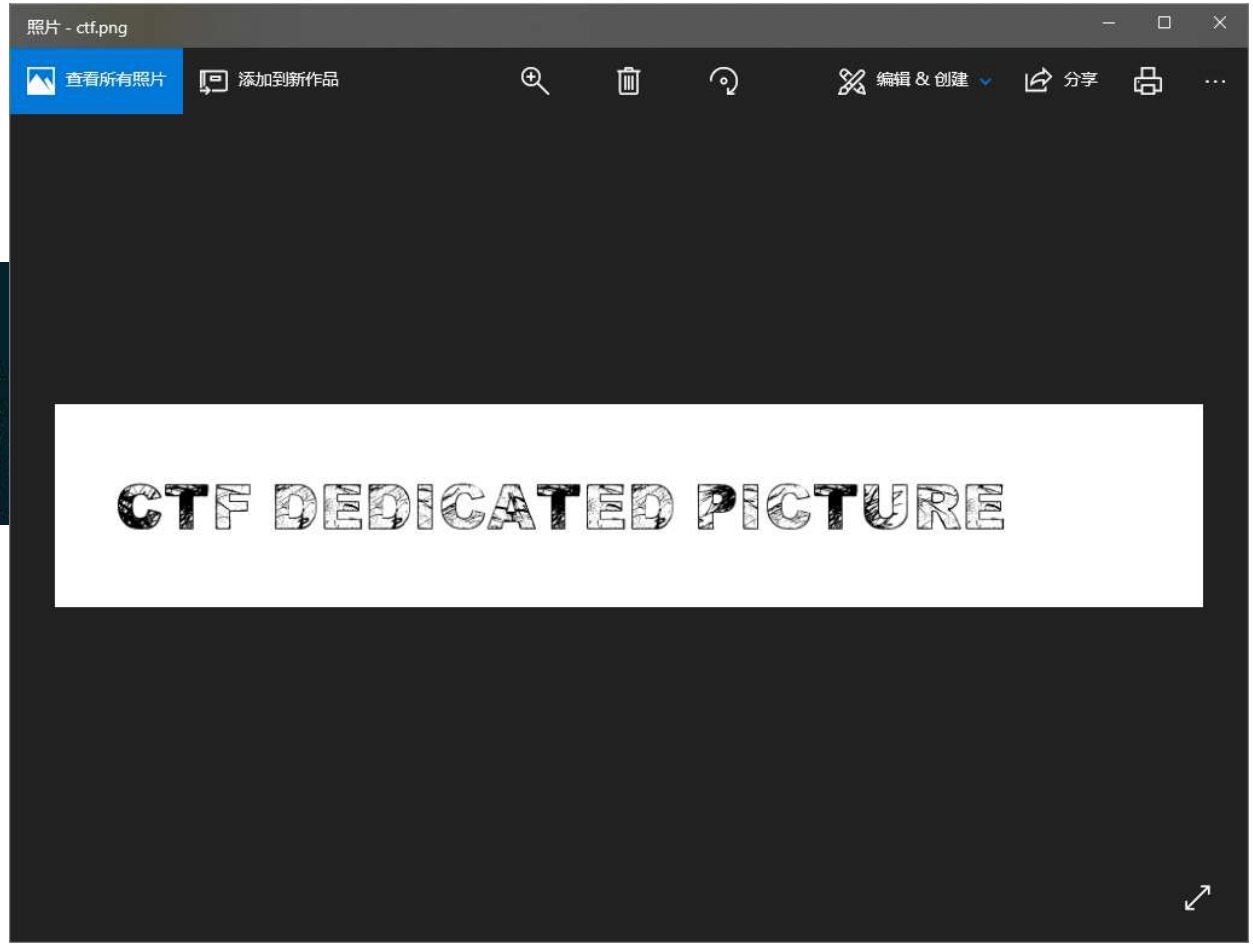
数据块符号	数据块名称	多数据块	可选否	位置限制
IHDR	文件头数据块	否	否	第一块
cHRM	基色和白色点数据块	否	是	在PLTE和IDAT之前
gAMA	图像γ数据块	否	是	在PLTE和IDAT之前
sBIT	样本有效位数据块	否	是	在PLTE和IDAT之前
PLTE	调色板数据块	否	是	在IDAT之前
bKGD	背景颜色数据块	否	是	在PLTE之后IDAT之前
hIST	图像直方图数据块	否	是	在PLTE之后IDAT之前
tRNS	图像透明数据块	否	是	在PLTE之后IDAT之前
oFFs	(专用公共数据块)	否	是	在IDAT之前
pHYs	物理像素尺寸数据块	否	是	在IDAT之前
sCAL	(专用公共数据块)	否	是	在IDAT之前
IDAT	图像数据块	是	否	与其他IDAT连续
tIME	图像最后修改时间数据块	否	是	无限制
tEXt	文本信息数据块	是	是	无限制
zTXt	压缩文本数据块	是	是	无限制
fRAc	(专用公共数据块)	是	是	无限制
gIFg	(专用公共数据块)	是	是	无限制
gIFt	(专用公共数据块)	是	是	无限制
gIFx	(专用公共数据块)	是	是	无限制
IEND	图像结束数据	否	否	最后一个数据块

0x02:Forensic&Steg

PNG文件格式

Magic Number

```
xu@ubuntu ~/CTF file ctf.png
ctf.png: data
xu@ubuntu ~/CTF hexdump -C ctf.png|more
00000000  12 50 4e 47 0d 0a 1a 0a  00 00 00 0d 49 48 44
00000010  00 00 03 ca 00 00 00 ac  04 03 00 00 00 51 04
00000020  dd 00 00 00 1b 50 4c 54  45 ff ff ff 00 00 00
00000030  df df bf bf bf 9f 9f 9f  7f 7f 7f 3f 3f 3f 5f
00000040  5f 1f 1f 1f 23 dc 65 b6  00 00 18 47 49 44 41
```



0x02:Forensic&Steg

PNG文件格式

IHDR


```
root@7f8f233ee778:/data# pngcheck
Here.png    hight.png   sctf.png
root@7f8f233ee778:/data# pngcheck hight.png
hight.png  CRC error in chunk IHDR (computed ff5859cc, expected a3f8bdbb)
ERROR: hight.png
root@7f8f233ee778:/data#
```

0x02:Forensic&Steg

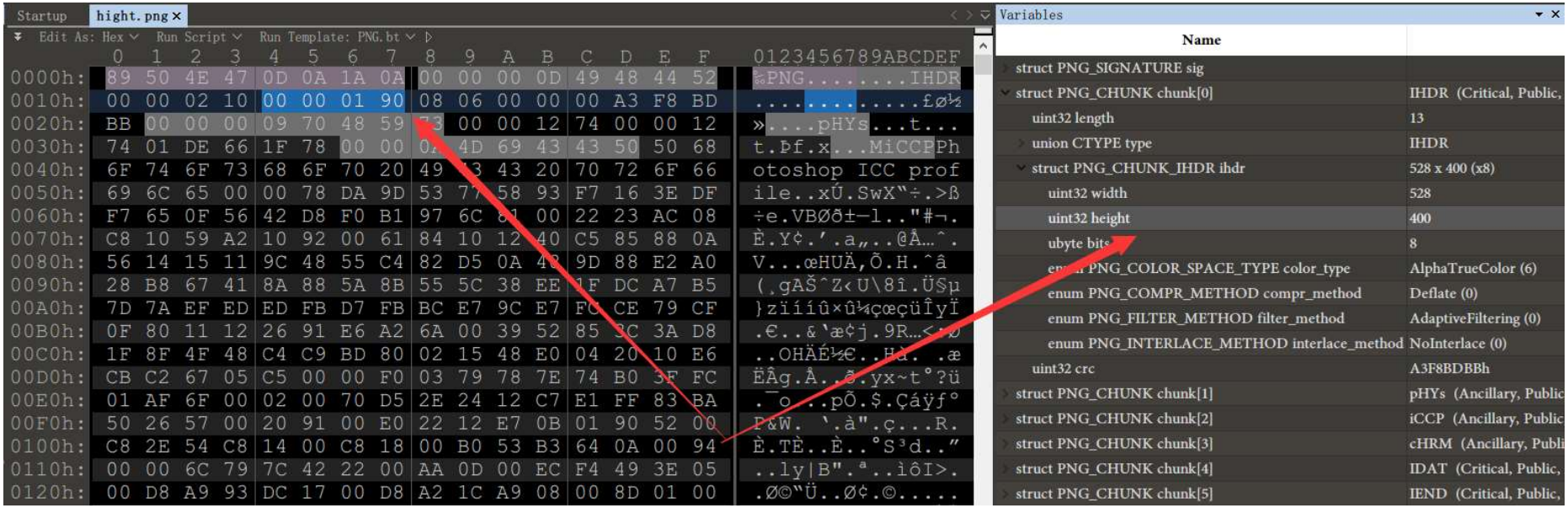
PNG文件格式

IHDR

0x02:Forensic&Steg

PNG文件格式

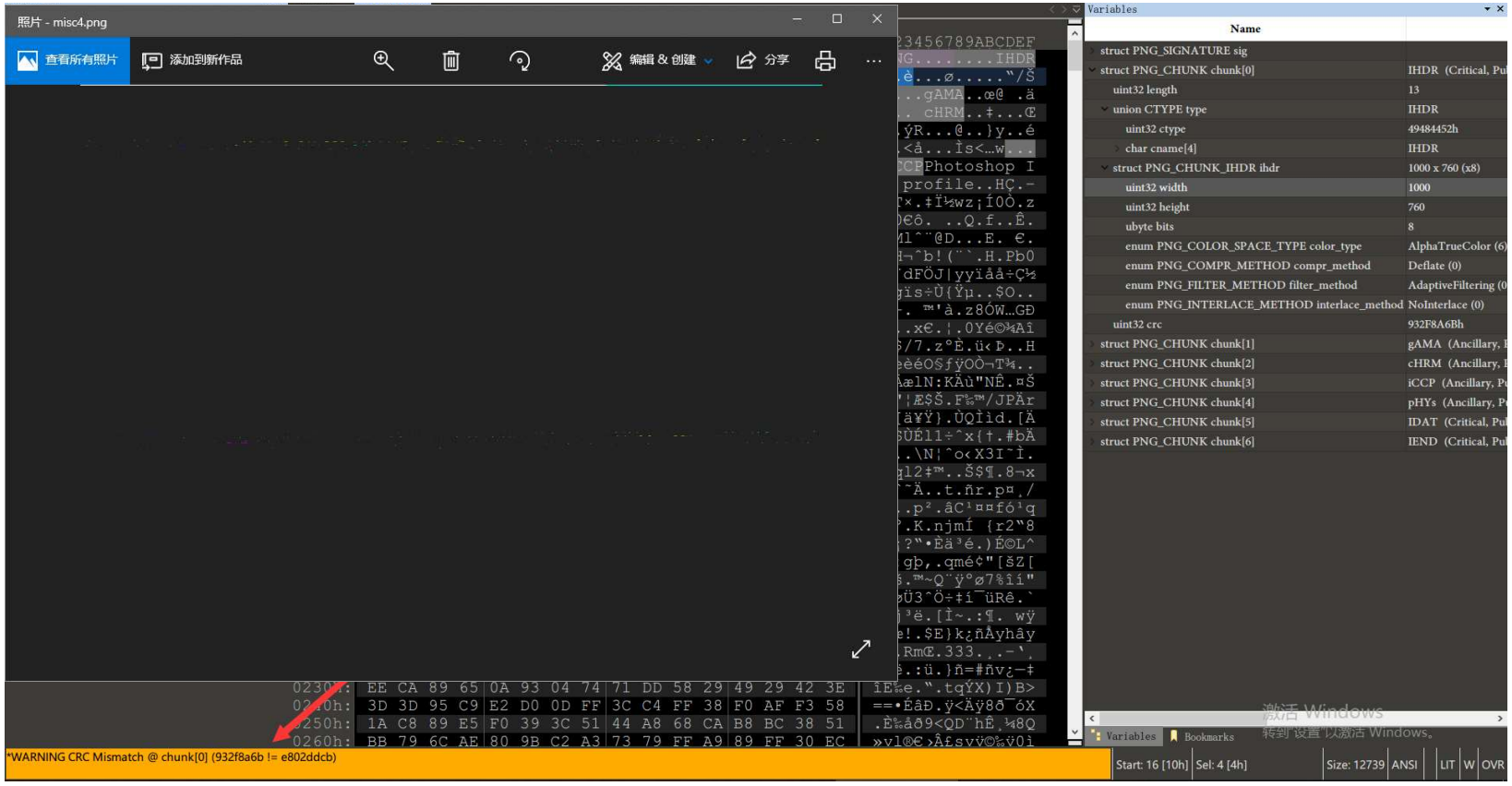
IHDR



0x02:Forensic&Steg

PNG文件格式

CRC



0x02:Forensic&Steg

PNG文件格式

CRC

CRC(cyclic redundancy check)域中的值是对Chunk Type Code域和Chunk Data域中的数据进行计算得到的。CRC具体算法定义在ISO 3309和ITU-T V.42中，其值按下面的CRC码生成多项式进行计算：
$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^{8}+x^{7}+x^{5}+x^{4}+x^{2}+x+1$

```
7
8  import os
9  import binascii
10 import struct
11
12
13 misc = open("misc4.png","rb").read()
14
15 for i in range(1024):
16     data = misc[12:16] + struct.pack('>i',i)+ misc[20:29]
17     crc32 = binascii.crc32(data) & 0xffffffff
18     if crc32 == 0x932f8a6b:
19         print i
20
21
```

0x02:Forensic&Steg

PNG文件格式

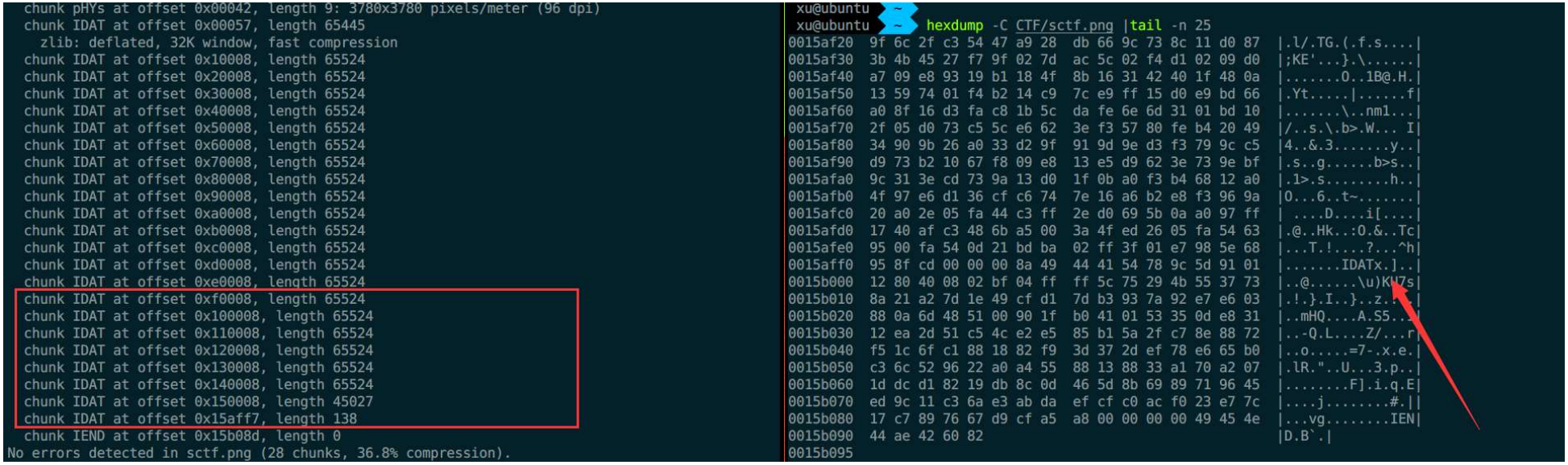
IDAT

```
root@7f8f233ee778:/data# pngcheck -v sctf.png
File: sctf.png (1421461 bytes)
  chunk IHDR at offset 0x0000c, length 13
    1000 x 562 image, 32-bit RGB+alpha, non-interlaced
  chunk sRGB at offset 0x00025, length 1
    rendering intent = perceptual
  chunk gAMA at offset 0x00032, length 4: 0.45455
  chunk pHYs at offset 0x00042, length 9: 3780x3780 pixels/meter (96 dpi)
  chunk IDAT at offset 0x00057, length 65445
    zlib: deflated, 32K window, fast compression
  chunk IDAT at offset 0x10008, length 65524
  chunk IDAT at offset 0x20008, length 65524
  chunk IDAT at offset 0x30008, length 65524
  chunk IDAT at offset 0x40008, length 65524
  chunk IDAT at offset 0x50008, length 65524
  chunk IDAT at offset 0x60008, length 65524
  chunk IDAT at offset 0x70008, length 65524
  chunk IDAT at offset 0x80008, length 65524
  chunk IDAT at offset 0x90008, length 65524
  chunk IDAT at offset 0xa0008, length 65524
  chunk IDAT at offset 0xb0008, length 65524
  chunk IDAT at offset 0xc0008, length 65524
  chunk IDAT at offset 0xd0008, length 65524
  chunk IDAT at offset 0xe0008, length 65524
  chunk IDAT at offset 0xf0008, length 65524
  chunk IDAT at offset 0x100008, length 65524
  chunk IDAT at offset 0x110008, length 65524
  chunk IDAT at offset 0x120008, length 65524
  chunk IDAT at offset 0x130008, length 65524
  chunk IDAT at offset 0x140008, length 65524
  chunk IDAT at offset 0x150008, length 45027
  chunk IDAT at offset 0x15aff7, length 138
  chunk IEND at offset 0x15b08d, length 0
No errors detected in sctf.png (28 chunks, 36.8% compression).
```

0x02:Forensic&Steg

PNG文件格式

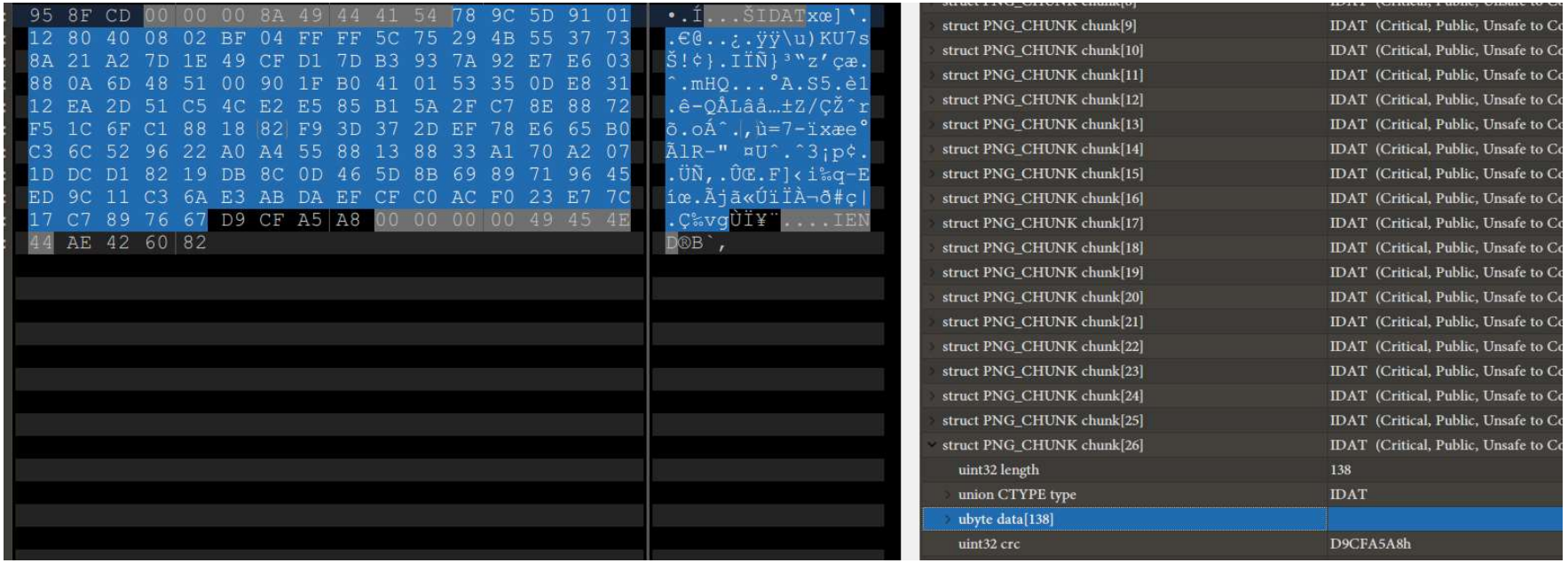
IDAT



0x02:Forensic&Steg

PNG文件格式

IDAT



LZ77: http://jazzlib.sourceforge.net/

0x02:Forensic&Steg

PNG文件格式

IDAT

LZ77: http://jazzlib.sourceforge.net/

```python
#! /usr/bin/env python
import zlib
import binascii
IDAT = "789C5D91011280400802BF04FFFF5C75294B5537738A21A27D1E49CFD17DB3937A9
#print IDAT
result = binascii.hexlify(zlib.decompress(IDAT))
print result

#print result.decode('hex')
```

```python
#!/usr/bin/env python
import Image
MAX = 25
pic = Image.new("RGB",(MAX, MAX))
str = "1111111000100001101111111100000101110010110100000110111010100000000010111011011101001000000001011101101110100
i=0
for y in range (0,MAX):
    for x in range (0,MAX):
        if(str[i] == '1'):
            pic.putpixel([x,y],(0, 0, 0))
        else:
            pic.putpixel([x,y],(255,255,255))
        i = i+1

pic.show()
pic.save("flag.png")
```

0x02:Forensic&Steg

PNG文件格式

IEND

```
root@7f8f233ee778:/data# ls
Here.png  sctf.png
root@7f8f233ee778:/data# binwalk he

General Error: Cannot open file : [Errno 2] No such file or directory: 'he'

root@7f8f233ee778:/data# binwalk Here.png

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             PNG image, 625 x 468, 8-bit/color RGBA, non-interlaced
62            0x3E            Zlib compressed data, default compression
300063        0x4941F         Zip archive data, at least v1.0 to extract, compressed size: 20,
uncompressed size: 20, name: flag.txt
300211        0x494B3         End of Zip archive

root@7f8f233ee778:/data#
root@7f8f233ee778:/data#
root@7f8f233ee778:/data#
root@7f8f233ee778:/data#
root@7f8f233ee778:/data#
root@7f8f233ee778:/data#
```

```
00049380  d9 83 37 ea 9f 42 83 ef  52 0c 47 1d ec 07 b6 cf  |..7..B..R.G.....|
00049390  c5 ad 36 71 dd 17 7d 06  ce 0b a8 ce 39 ef 78 b1  |..6q..}.....9.x.|
000493a0  fd dd f6 eb ea d6 b5 a0  ed 08 e9 d2 00 42 35 aa  |.............B5.|
000493b0  7c 7a 31 cf 50 c6 55 71  d0 d6 f7 58 48 0c 44 51  ||z1.P.Uq...XH.DQ|
000493c0  60 27 83 d0 14 e6 73 b7  cf 84 0b ba eb 7a cb 81  |`'....s......z..|
000493d0  ff 7f 6f c8 6c 21 39 d6  ce db f0 fd b5 e7 c5 cf  |..o.l!9.........|
000493e0  03 12 81 13 f2 04 83 f3  b6 5d d3 03 6a 9b 40 db  |.........]..j.@.|
000493f0  36 38 7b 15 b8 83 8c 1b  74 7d 75 55 49 c8 6c 04  |68{.....t}uUI.l.|
00049400  f3 a5 b4 33 c2 5c 0c 58  e4 ff 03 55 4f 26 73 88  |...3.\.X...UO&s.|
00049410  a9 41 81 00 00 00 00 49  45 4e 44 ae 42 60 82 50  |.A.....IEND.B`.P|
00049420  4b 03 04 0a 00 00 00 00  00 83 72 f5 4a 1c 5d a6  |K.........r.J.].|
00049430  45 14 00 00 00 14 00 00  00 08 00 00 00 66 6c 61  |E............fla|
00049440  67 2e 74 78 74 66 6c 61  67 7b 48 65 72 65 5f 68  |g.txtflag{Here_h|
00049450  40 76 33 5f 66 6c 61 67  7d 50 4b 01 02 3f 00 0a  |@v3_flag}PK..?..|
00049460  00 00 00 00 00 83 72 f5  4a 1c 5d a6 45 14 00 00  |......r.J.].E...|
00049470  00 14 00 00 00 08 00 24  00 00 00 00 00 00 00 20  |.......$....... |
00049480  00 00 00 00 00 00 00 66  6c 61 67 2e 74 78 74 0a  |.......flag.txt.|
00049490  00 20 00 00 00 00 00 01  00 18 00 e1 ff f8 64 e9  |. ............d.|
000494a0  01 d3 01 46 ed fd 4f e9  01 d3 01 46 ed fd 4f e9  |...F..O....F..O.|
000494b0  01 d3 01 50 4b 05 06 00  00 00 00 01 00 01 00 5a  |...PK.........Z|
000494c0  00 00 00 3a 00 00 00 00  00                       |...:.....|
000494c9
```

0x02:Forensic&Steg

PNG文件格式

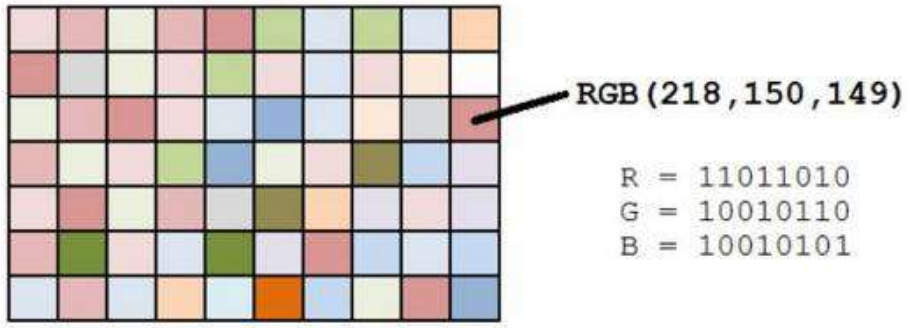
LSB

最低有效位 (Least Significant Bit)

最高位对图像影响最大，最低位对图像影响最小，故称为最低有效位

PNG由RGB三原色构成，每种颜色占用8bits，即取值为0x00~0xFF，即有256种颜色

Bmp灰度值



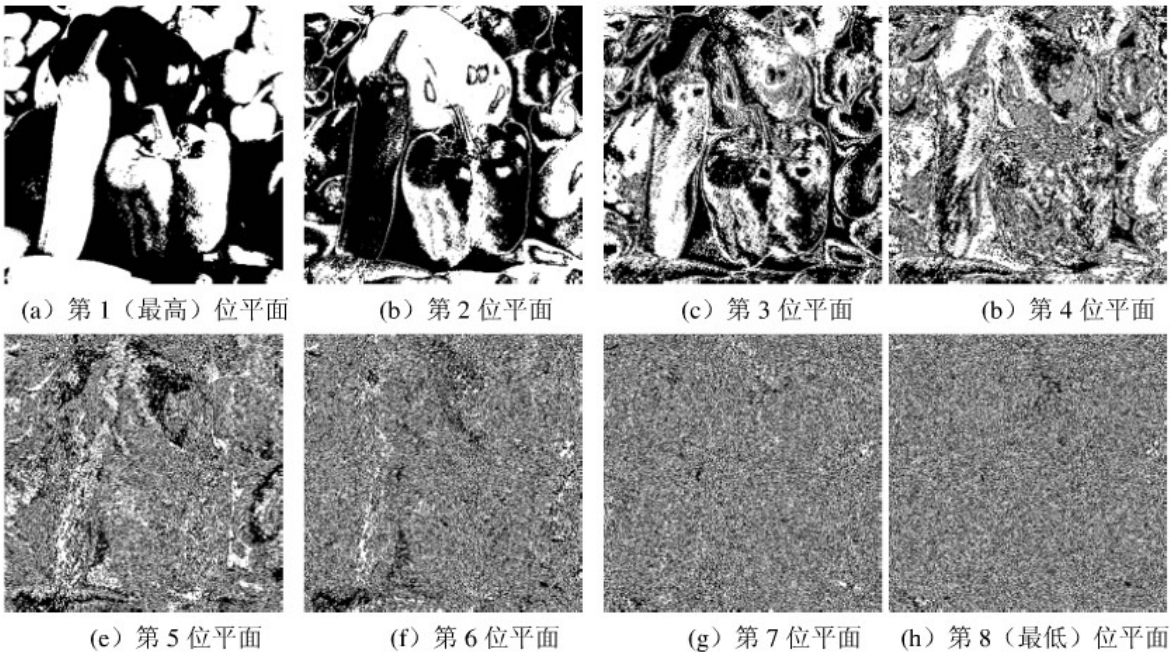
0x02:Forensic&Steg

PNG文件格式

LSB

Stegsolve:

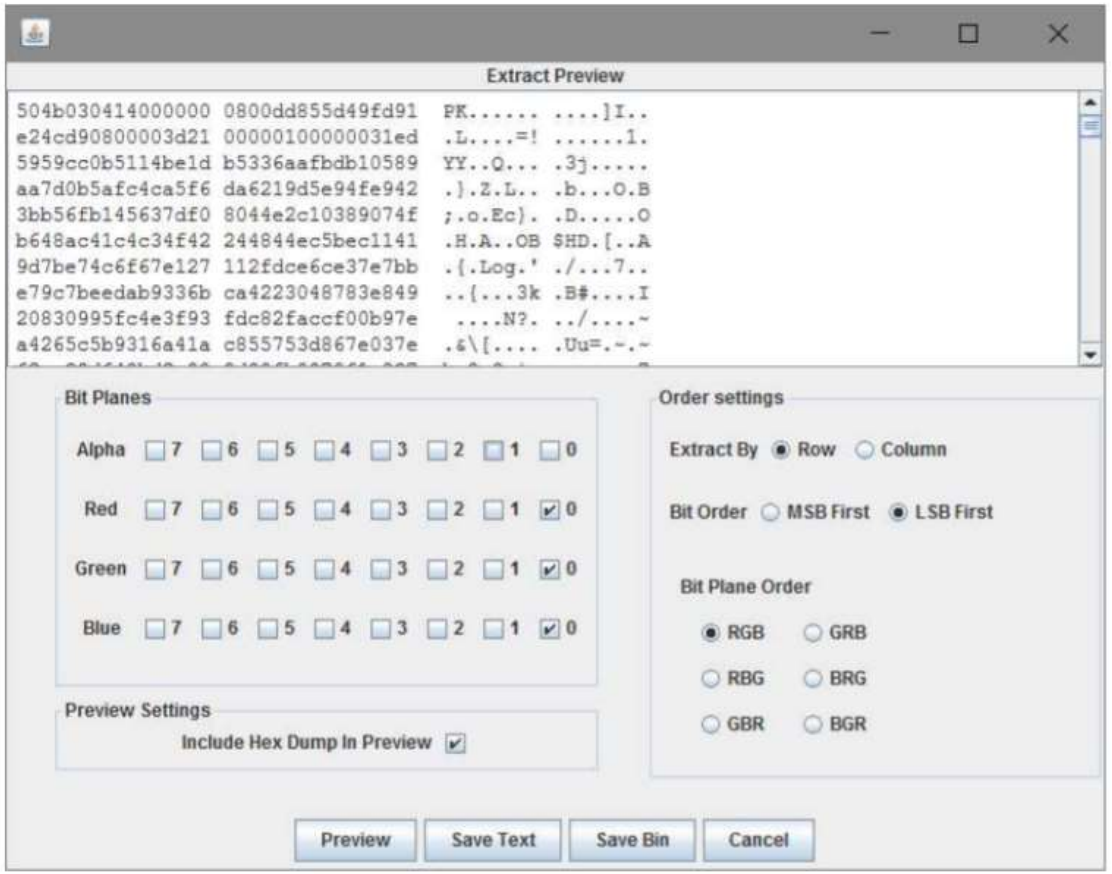
图片通道查看器
图片隐写必备



0x02:Forensic&Steg

PNG文件格式

LSB



0x02:Forensic&Steg

PNG文件格式

LSB

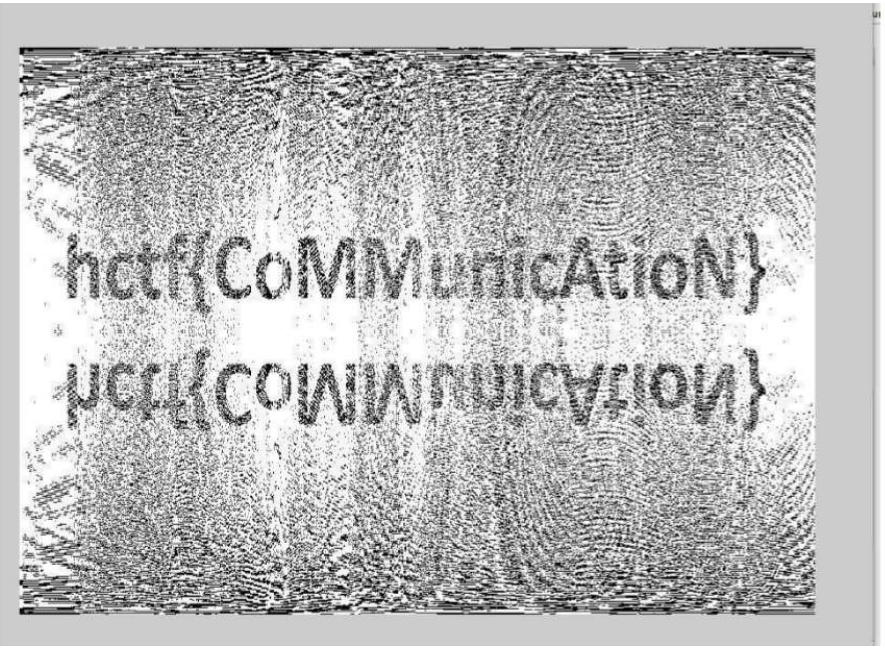
```
for y in xrange(height):
    for x in xrange(width):

        pixel = list(im.getpixel((x, y)))

        for i in xrange(3):
            count = pixel[i]%2

            if len(flag) == 0:
                break

            if count == int(flag.pop()):
                continue

            if count == 0:
                pixel[i]+=1

            elif count == 1:
                pixel[i]-=1

        pic.putpixel([x, y],tuple(pixel))
```

0x02:Forensic&Steg

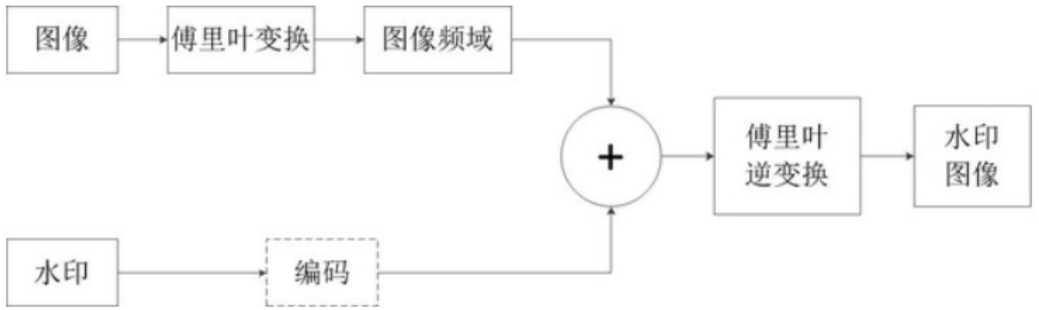
PNG文件格式

频域手段添加数字盲水印

https://www.zhihu.com/question/50735753




```
IM = imread('shimakaze.bmp');
IM_FFT = fft(IM);
imshow(real(uint8(IM_FFT)));
```



0x02:Forensic&Steg

JPG文件格式

标记码+压缩数据

标记码： 由两个字节构成，第一个字节是固定值0xFF，后一个字节则根据不同意义有不同数值
压缩数据：前两个字节保存整个段的长度，包括这两个字节

```
SOI 0xD8 图像开始
APP0 0xE0 应用程序保留标记0
APPn 0xE1 - 0xEF 应用程序保留标记n(n=1～15)
DQT 0xDB 量化表(Define Quantization Table)
SOF0 0xC0 帧开始(Start Of Frame)
DHT 0xC4 定义Huffman表(Define Huffman Table)
DRI 0XDD 定义差分编码累计复位的间隔(Define Restart Interval)
SOS 0xDA 扫描开始(Start Of Scan)
EOI 0xD9 图像结束
```

段标识	1	FF	每个新段的开始标识
段类型	1		称作“标记码”
段长度	2		包括段内容和段长度本身,不包括段标识和段类型
	PS：0x0000~0xFFFF 占4个字节		
段内容			≤65533字节

0x02:Forensic&Steg

JPG文件格式

利用文件格式隐藏信息：

插入法：

1.文件尾部插入

储存在APP1数据区中

MagicEXIF

2.每段开始前：

COM注释：
FF 开始标记
FE COM注释标记符
00 02 总长度
11 内容

0x02:Forensic&Steg

JPG文件格式

JPHIDE 隐写

stegdetect 检测隐写工具

```
D:\Tool-CTF\Tools\CTF□□□□\□□д\□□дstegdetect\stegdetect\stegdetect
λ stegdetect.exe -tjopi -s 1000.0 Pcat.jpg
Pcat.jpg : jphide(***)

D:\Tool-CTF\Tools\CTF□□□□\□□д\□□дstegdetect\stegdetect\stegdetect
λ stegdetect.exe -tjopi -s 1000.0 Outguess.jpg
Outguess.jpg : outguess(old)(***)
```

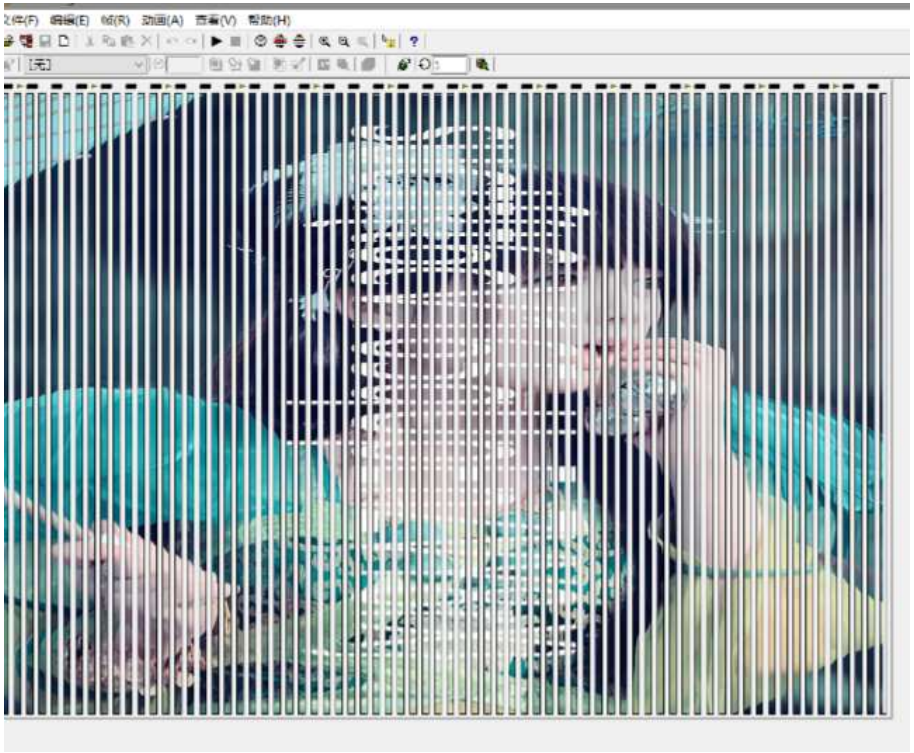
-t 设置要检测哪些隐写工具（默认检测jopi），可设置的选项如下：
-j 检测图像中的信息是否是用jsteg嵌入的。
-o 检测图像中的信息是否是用outguess嵌入的。
-p 检测图像中的信息是否是用jphide嵌入的。
-i 检测图像中的信息是否是用invisible secrets嵌入的。

0x02:Forensic&Steg

GIF类

Gif在线编辑器： http://ezgif.com/split

- 文件头
- 时间轴
- 空间域



0x02:Forensic&Steg

GIF类

时间轴 ：XMAN-2017:100.gif

- 文件头
- 时间轴
- 空间域

```
$ identify -format "%s %T \n" 100.gif
0 66
1 66
2 20
3 10
4 20
5 10
6 10
7 20
8 20
9 20
10 20
11 10
12 20
13 20
14 10
15 10
```

0x02:Forensic&Steg

GIF类

Gif在线编辑器： http://ezgif.com/split

- 文件头
- 时间轴
- 空间域

0x04:Steg

音频类：

Adobe audition/Audacity：频谱，波形的处理

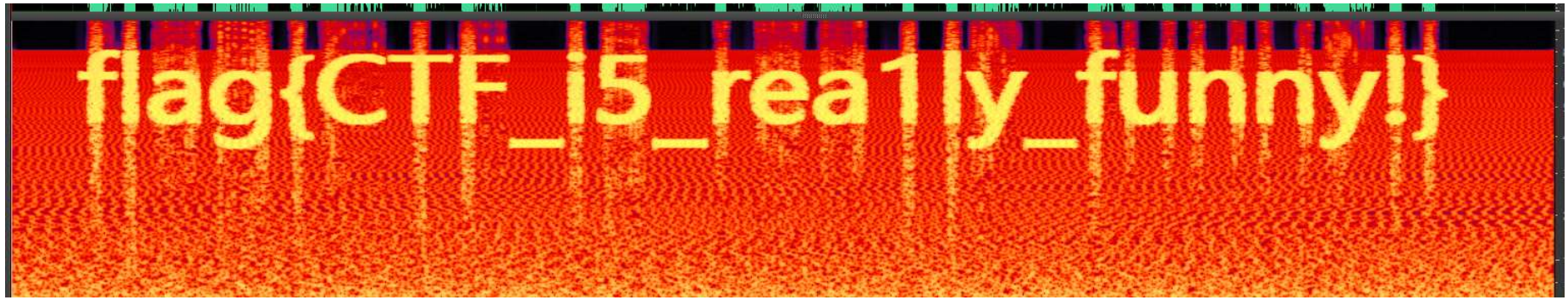
MP3Stego：敏感数据压缩隐藏到MP3音乐文件中

- 频谱
- 波形
- Other

0x04:Steg

音频类：

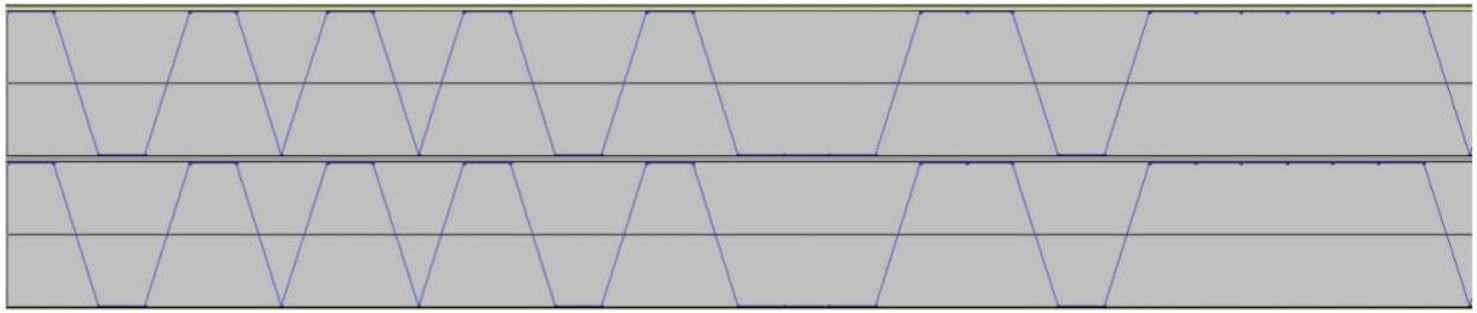
频谱

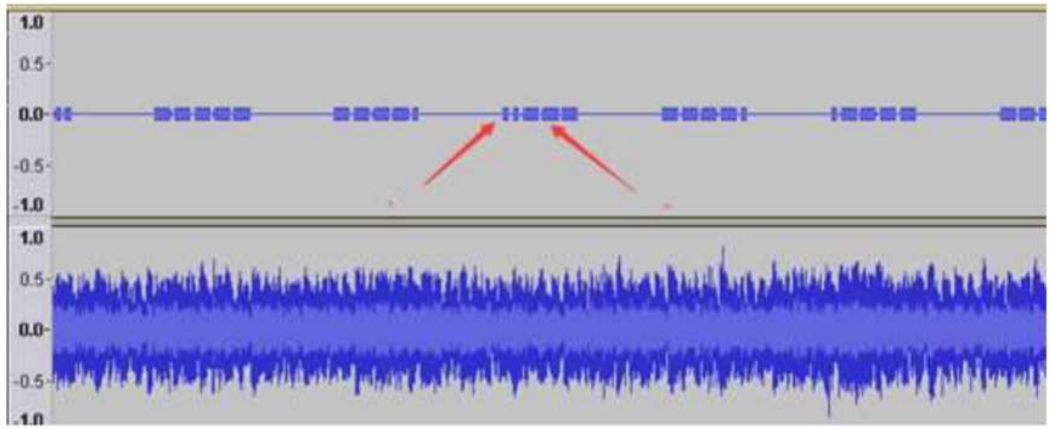


0x04:Steg

音频类：

波形





0x04:Steg

音频类：

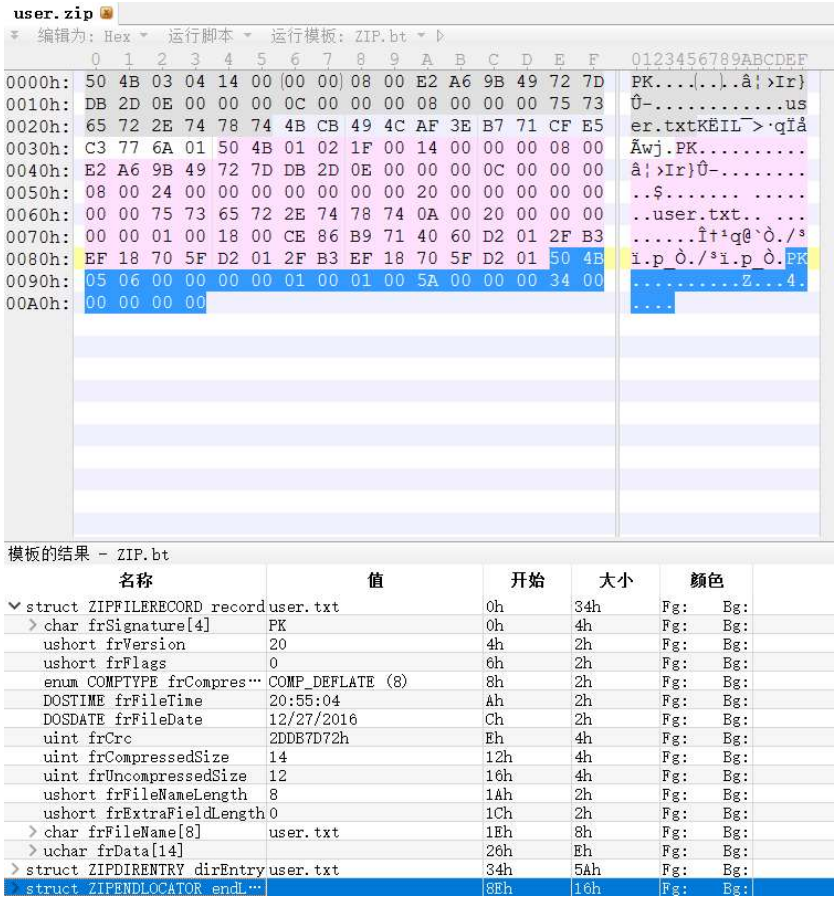
Other

- Mp3Stego　　http://cea.ceaj.org/CN/article/downloadArticleFile.do?attachType=PDF&id=33065
    - ISCC-2016: Music Never Sleep
- SlientEye

0x02:Forensic&Steg

ZIP文件格式：

组成：
-压缩源文件数据区
-压缩源文件目录区
-压缩源文件目录结束标志



0x02:Forensic&Steg

ZIP文件格式

压缩源文件数据区：文件头+ 文件数据 + 数据描述符
记录所有压缩文件的内容信息

文件头结构说明：右图

文件数据：对应压缩文件源数据

数据描述符：

Header			
Offset	Bytes	Description	译
0	4	Local file header signature = 0x04034b50 (read as a little-endian number)	文件头标识，值固定(0x04034b50)
4	2	Version needed to extract (minimum)	解压文件所需 pkware最低 版本

Data descriptor			
Offset	**Bytes**	**Description**[18]	**译**
0	4	Local file header signature = 0x08074b50	本地header标记
4	4	CRC-32	CRC-32
8	4	Compressed size	压缩后大小
12	4	Uncompressed size	非压缩的大小
22	4	Uncompressed size	非压缩的大小
26	2	File name length (*n*)	文件名长度
28	2	Extra field length (*m*)	扩展区长度
30	n	File name	文件名
30+n	m	Extra field	扩展区

0x02:Forensic&Steg

ZIP文件格式

压缩的目录源数据:

每个压缩目录元数据
对应原来的一个目录

Central directory file header			
Offset	**Bytes**	**Description**[18]	**译**
0	4	Central directory file header signature = 0x02014b50	核心目录文件header标识=（0x02014b50）
4	2	Version made by	压缩所用的pkware版本
6	2	Version needed to extract (minimum)	解压所需pkware的最低版本
8	2	General purpose bit flag	通用位标记
10	2	Compression method	压缩方法
12	2	File last modification time	文件最后修改时间
14	2	File last modification date	文件最后修改日期
16	4	CRC-32	CRC-32算法
20	4	Compressed size	压缩后大小
24	4	Uncompressed size	未压缩的大小
28	2	File name length (*n*)	文件名长度
30	2	Extra field length (*m*)	扩展域长度
32	2	File comment length (*k*)	文件注释长度
34	2	Disk number where file starts	文件开始位置的磁盘编号
36	2	Internal file attributes	内部文件属性
38	4	External file attributes	外部文件属性
42	4	Relative offset of local file header. This is the number of bytes between the start of the first disk on which the file occurs, and the start of the local file header. This allows software reading the central directory to locate the position of the file inside the ZIP file.	本地文件header的相对位移。
46	*n*	File name	目录文件名
46+*n*	*m*	Extra field	扩展域
46+*n*+*m*	*k*	File comment	文件注释内容

0x02:Forensic&Steg

ZIP文件格式

目录结束标识结构:

标记压缩的目录数据的结束

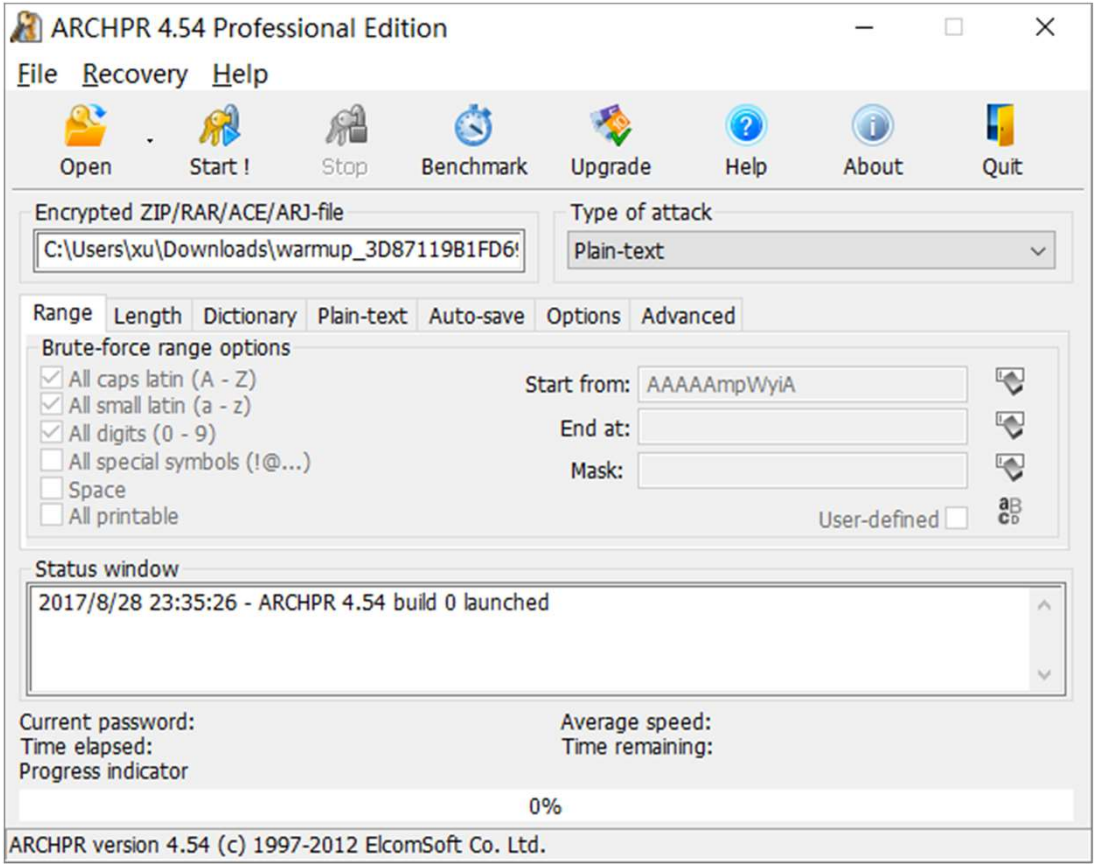
End of central directory record			
Offset	**Bytes**	**Description**[18]	译
0	4	End of central directory signature = 0x06054b50	核心目录结束标记（0x06054b50）
4	2	Number of this disk	当前磁盘编号
6	2	Disk where central directory starts	核心目录开始位置的磁盘编号
8	2	Number of central directory records on this disk	该磁盘上所记录的核心目录数量
10	2	Total number of central directory records	核心目录结构总数
12	4	Size of central directory (bytes)	核心目录的大小
16	4	Offset of start of central directory, relative to start of archive	核心目录开始位置相对于archive开始的位移
20	2	Comment length (*n*)	注释长度
22	*n*	Comment	注释内容

0x02:Forensic&Steg

ZIP文件格式

爆破

- ARCHPR
- fcrackzip

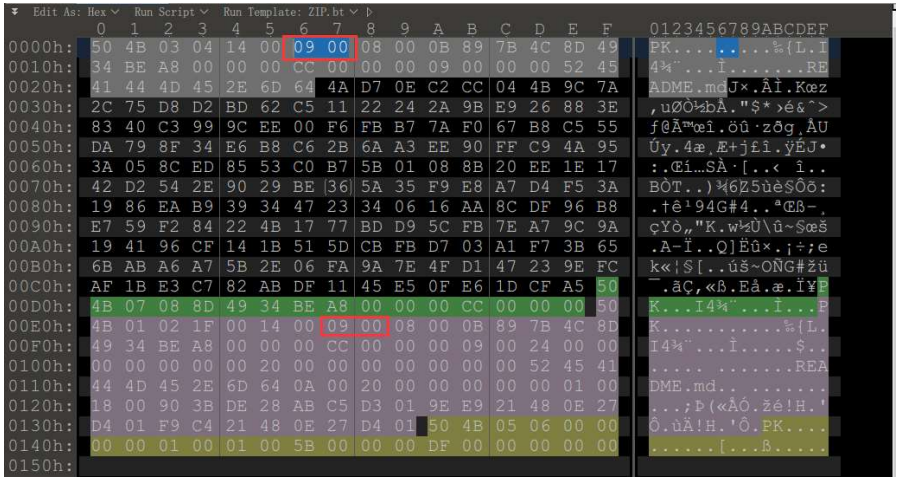


0x02:Forensic&Steg

ZIP文件格式

伪加密

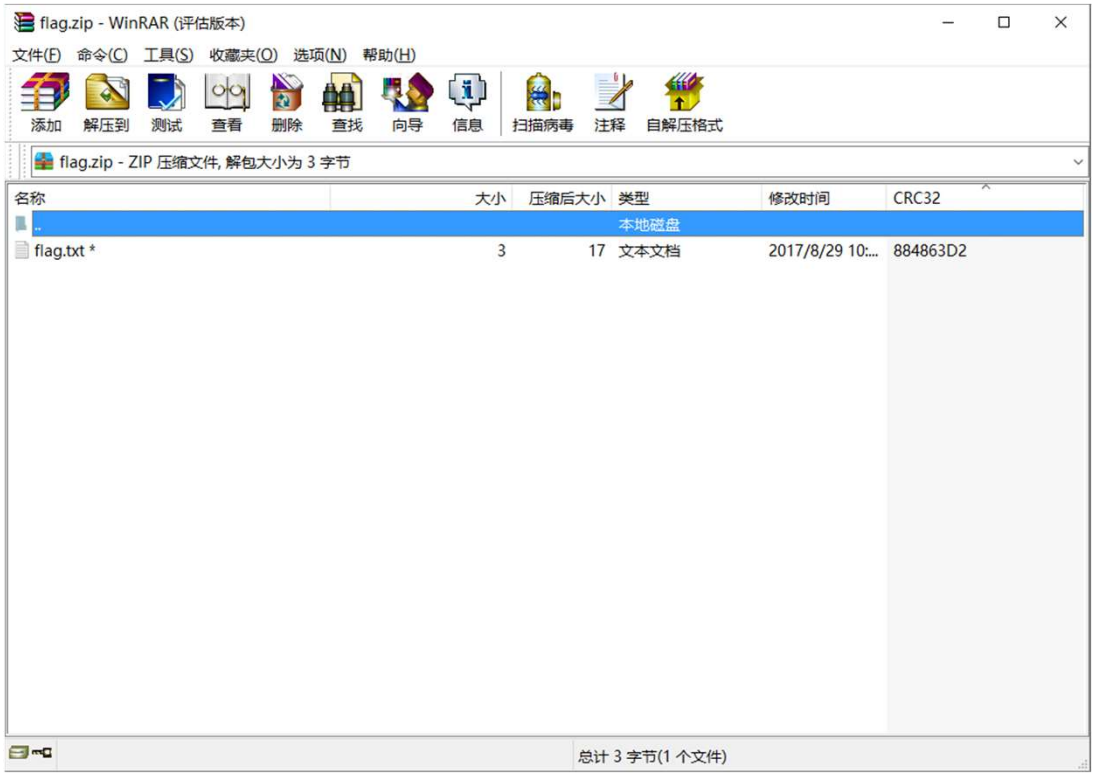
修改加密标志位



0x02:Forensic&Steg

ZIP文件格式

CRC32碰撞



0x02:Forensic&Steg

ZIP文件格式

明文攻击

http://www.unix-ag.uni-kl.de/~conrad/krypto/pkcrack.html

0x02:Forensic&Steg

流量分析

- 文件修复： pcapfix
- 协议分析： wireshark
- 数据提取： tshark

0x02:Forensic&Steg

流量分析

文件格式

Global Header	Packet Header	Packet Data	Packet Header	Packet Data	Packet Header	Packet Data	...

Global Header

```
typedef struct pcap_hdr_s {
        guint32 magic_number;   /* magic number */
        guint16 version_major;  /* major version number */
        guint16 version_minor;  /* minor version number */
        gint32  thiszone;       /* GMT to local correction */
        guint32 sigfigs;        /* accuracy of timestamps */
        guint32 snaplen;        /* max length of captured packets, in octets */
        guint32 network;        /* data link type */
} pcap_hdr_t;
```

0x02:Forensic&Steg

流量分析

文件格式

Packet Header

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;         /* timestamp seconds */
        guint32 ts_usec;        /* timestamp microseconds */
        guint32 incl_len;       /* number of octets of packet saved in file */
        guint32 orig_len;       /* actual length of packet */
} pcaprec_hdr_t;
```

0x02:Forensic&Steg

流量分析

常见协议

- Http
- Https
- DNS
- FTP

```
Acknowledgment number: 46    (relative ack number)
Header Length: 20 bytes
Flags: 0x030 (ACK, URG)
   000. .... .... = Reserved: Not set
   ...0 .... .... = Nonce: Not set
   .... 0... .... = Congestion Window Reduced (CWR): Not set
   .... .0.. .... = ECN-Echo: Not set
   .... ..1. .... = Urgent: Set
   .... ...1 .... = Acknowledgment: Set
   .... .... 0... = Push: Not set
   .... .... .0.. = Reset: Not set
   .... .... ..0. = Syn: Not set
   .... .... ...0 = Fin: Not set
   [TCP Flags: ·····UA····]
Window size value: 8192
[Calculated window size: 8192]
[Window size scaling factor: -2 (no window scaling used)]
Checksum: 0x5917 [unverified]
[Checksum Status: Unverified]
Urgent pointer: 67
```

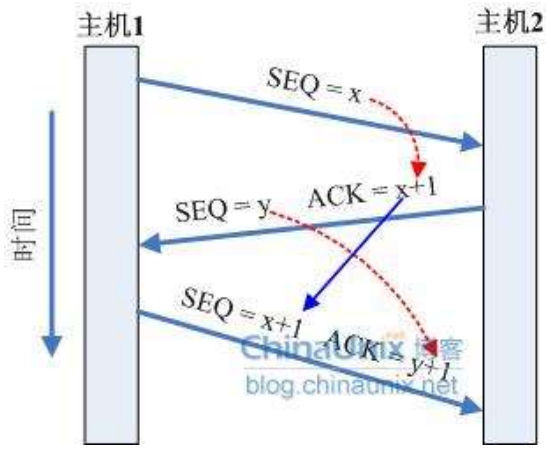
0x02:Forensic&Steg

流量分析

0x02:Forensic&Steg

流量分析

隐藏数据：

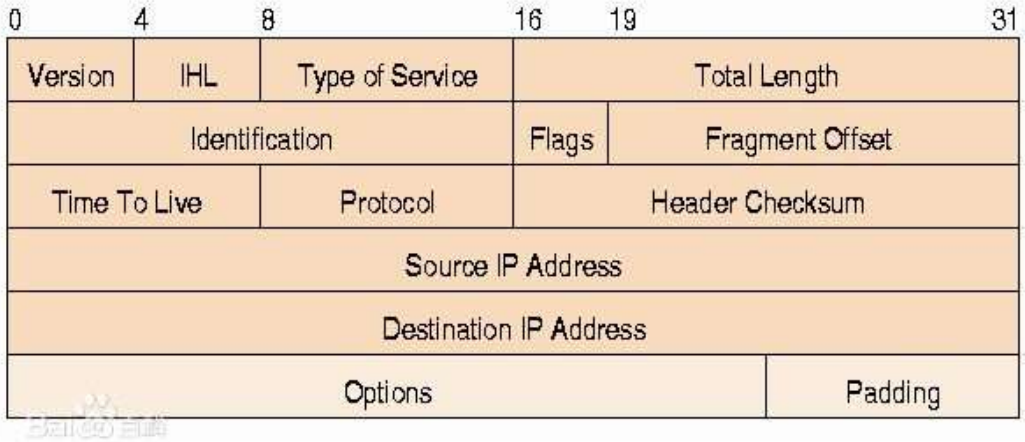


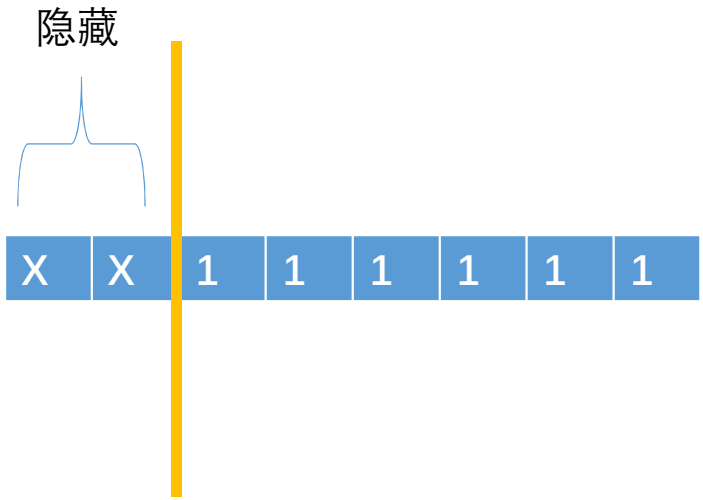

```
Transmission Control Protocol, Src Port: 49200, Dst Port: 80, Seq: 693, Ack: 390, L
    Source Port: 49200
    Destination Port: 80
    [Stream index: 8]
    [TCP Segment Len: 541]
    Sequence number: 693    (relative sequence number)
    [Next sequence number: 1234    (relative sequence number)]
    Acknowledgment number: 390    (relative ack number)
    Header Length: 20 bytes
  > Flags: 0x018 (PSH, ACK)
    Window size value: 16327
    [Calculated window size: 65308]
    [Window size scaling factor: 4]
```

0x02:Forensic&Steg

流量分析

隐藏数据：

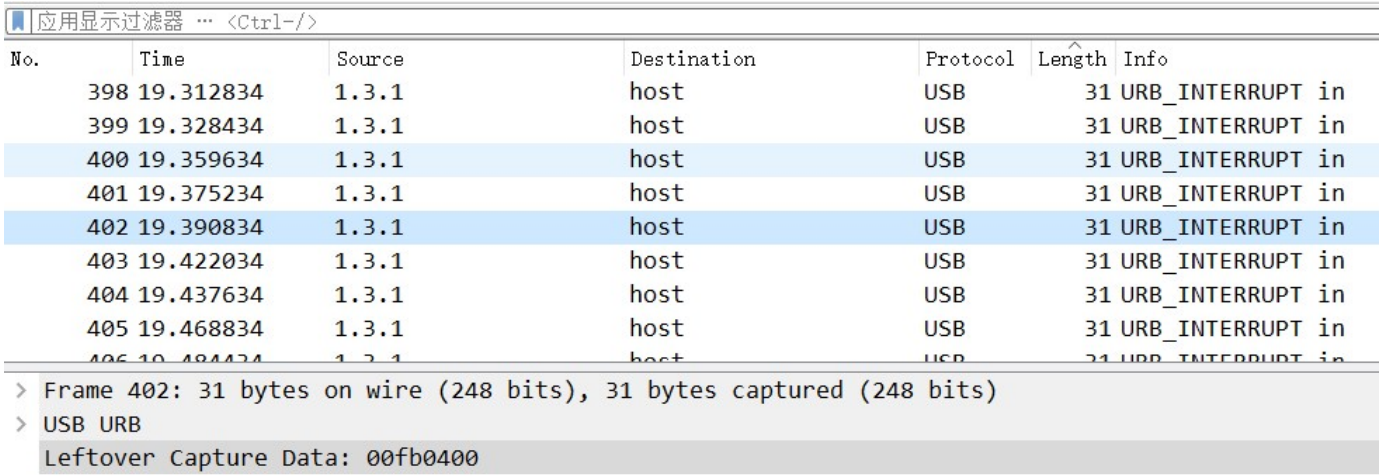




0x02:Forensic&Steg

流量分析

Google CTF 2016 Forensic-200

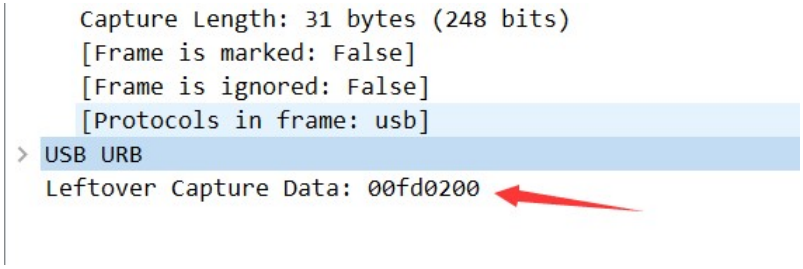


0x02:Forensic&Steg

流量分析

Google CTF 2016 Forensic-200

USB 鼠标的数据为 4 个字节，第四个字节一般不会用到。
第一个字节为鼠标的状态，前 5 个 bit 不用管，第 6 个 bit 是鼠标中键按下的状态，第 7 个 Bit 是鼠标右键的状态，最后一个是左键
第二个字节为 X 的移动，signed int ，第三个字节为 Y 的移动，signed int
题目中的第一个字节不是 0 就是 1，说明只用到了鼠标左键，我们写脚本将鼠标移动的图片画出来即可获得 Flag



0x02:Forensic&Steg

流量分析

铁人三项 数据分析赛

- 黑客最终获得了什么用户名
- 黑客最终获得了什么密码
- 黑客修改了什么文件
- 黑客使用菜刀的完整连接地址
- 黑客使用菜刀的连接密码
- 黑客的查看的第一个文件目录是什么

0x02:Forensic&Steg

流量分析

铁人三项 数据分析赛

- *172.16* *rer*
- *172.16* *o Referrer*
- *101.36* *No Referr*

```
[219.239.105.18:5109] -- -- --> [172.16.61.210:80]
POST /index.php?m=search HTTP/1.1
X-Forwarded-For: 18.124.107.251
Referer: http://118.194.196.232
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 118.194.196.232:8083
Content-Length: 692
Connection: Close
Cache-Control: no-cache
Cookie: kNmvb_admin_username=34baUQUGAAQDVAMBVgYMUFUDBVwFVVJRDgZaUgARWUhUDEU;
kNmvb_siteid=e4b7UlQJCAACAAgBAwQPUgFSAAlVVAgHCgVZA1QD; kNmvb_userid=e4b7UlQJCAACAAgBAwQPUgFSAAlVVAgHCgVZA1QD;
kNmvb_admin_email=1578VgQCCFIJBFIFBlRXD19fAQJQUlFWUlQDBgIABFZQVgBQAXIJUAoaAVpb;
kNmvb_sys_lang=ab28BAIIU1EHBAVSAgcOVwMCUQUHVwNXXgsBVAhJXUlUVg;
wUsRn_admin_username=eb3cAVUIVQMFBlQGUQRcU1ULUANeBlVQBgMBVQYTDltA; wUsRn_siteid=cc1OUgkIUgEJBAgGAwAEUwEGX1JXAFRTVAMCBVUF;
wUsRn_userid=cc1OUgkIUgEJBAgGAwAEUwEGX1JXAFRTVAMCBVUF;
wUsRn_admin_email=b40bBVJSAFNRUQQBU1FeBQdXAgEFBlECDgRRU1NGAAJDB3EGUFEETQIOWQ;
wUsRn_sys_lang=f2a4BlUBUgQFBwlSBg1TVFRRVFJdUgUFUVRdUFVKW0tVCA

chopper=%40eval%01%28base64_decode%28%24_POST%5Bz0%5D%29%29%3B&z0=QGluaV9zZXQoImRpc3BsYXlfZXJyb3JzIiwiMCIpOOBzZXRfdGltZV9saWl
pdCgwKTsAc2V0X21hZ2ljX3F1b3Rlc19ydW50aW1lKDApO2VjaG8oIi0%2BfCIpOzskRD1kaXJuYW1lKCRfU0VSVkVSWyJTQ1JJUFRfRklMRU5BTUUiXSk7aWYoJ
Q9PSIiKSREFM9pcm5hbWUoJF9TRVJWRVJbIlBBVEhfVFJBTlNMQVRFRCJdKTskUjOieyREfVx0IjtpZihzdWJzdHIoJEQsMCwxKSE9Ii8iKXtmb3JlYWNoKHJhbmd
lKCJBIiwiWiIpIGFzICRMKWlmKGlzX2RpcigieyRMfToiKSkkUi49InskTH06Ijt9JFIuPSJcdCI7JHU9KGZ1bmN0aW9uX2V4aXN0cygncG9zaXhfZ2V0ZWdpZCcp
KT9AcG9zaXhfZ2V0cHd1aWQoQHBvc2l4X2dldGV1aWQoKSk6Jyc7JHVzcj0oJHUpPyR1WyduYW1lJ106QGdldF9jdXJyZW50X3VzZXIoKTskUi49cGhwX3VuYW1lK
Ck7JFIuPSIoeyR1c3J9KSI7cHJpbnQgJFI7O2VjaG8oInw8LSIpO2RpZSgpOw%3D%3D

HTTP/1.1 200 OK
Date: Wed, 10 Aug 2016 06:15:10 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.3.3
Content-Length: 119
```

0x02:Forensic&Steg

流量分析

非常见协议

USB

.....

```
> Frame 70: 35 bytes on wire (280 bits), 35 bytes captured (280 bits)
> USB URB
  Leftover Capture Data: 00fb0200fbff0200
```

```
    Capture Length: 31 bytes (248 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: usb]
> USB URB
  Leftover Capture Data: 00fd0200
```

0x02:Forensic&Steg

流量分析

数据提取

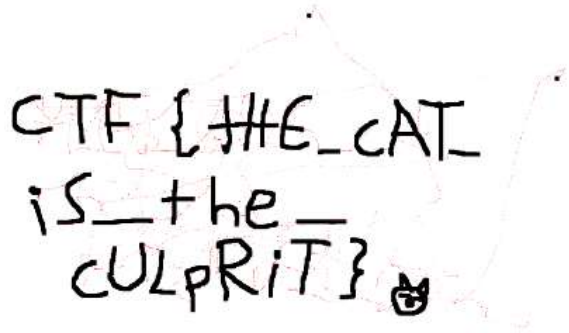
- 流重组：Follow TCP Stream
- File-->Export Objects-->HTTP
- 手工提取：
  - Python：pcapy
  - Tshark：
    - -r **.pcap –Y ** -T fields –e ** |

```
tshark -r fore2.pcap -Y 'usb.capdata and usb.device_address==3' -T fields -e usb.capdata > raw
```

0x02:Forensic&Steg

Google CTF 2016 Forensic-200

文件名：capture.pcapng



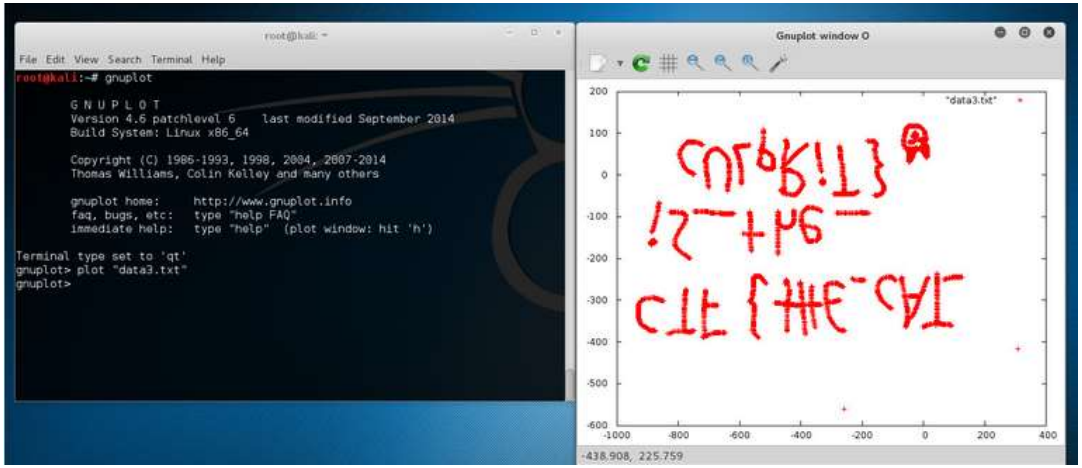
键盘的流量包，投影仪的流量包

```python
import struct
from PIL import Image
import dpkt
import binascii

INIT_X, INIT_Y = 1000, 1000

def compliment(h):
    i = int(h, 16)
    return i - ((0x80 & i) << 1)

def print_map(pcap, device):
    picture = Image.new("RGB", (2200, 2500), "white")
    pixels = picture.load()

    x, y = INIT_X, INIT_Y

    for ts, buf in pcap:
        mouse_data = buf[27:32]
        mdstr = binascii.hexlify(mouse_data);
        ml = len(mdstr)
        if ml != 8:
            continue
        bits = int(mdstr[0] + mdstr[1],base=16)
        bx = compliment(mdstr[2] + mdstr[3])
        by = compliment(mdstr[4] + mdstr[5])

        if bx > 20:
            continue
        if by > 20:
            continue

        status = bits
        x = x + bx
        y = y + by

        if (status == 1):
            for i in range(-5, 5):
                for j in range(-5, 5):
                    pixels[x + i , y + j] = (0, 0, 0, 0)
        else:
            pixels[x, y] = (255, 0, 0, 0)
    picture.save("out.png", "PNG")

if __name__ == "__main__":

    f = open("1.pcap", "rb")
    pcap = dpkt.pcap.Reader(f)

    print_map(pcap, 3)
    f.close()
```

0x02:Forensic&Steg

Google CTF 2016 Forensic-200

文件名：capture.pcapng



what@kali:/tmp$ **tshark -r capture.pcapng -T fields -e usb.capdata > data2.txt**
what@kali:/tmp$ **tail -n 20 data.txt**

```
00:ff:00:00
00:ff:00:00
00:ff:00:00
00:fd:00:00
00:ff:00:00
00:ff:00:00
00:fe:ff:00
00:fd:00:00
```

root@kali:~# **awk -F: 'function comp(v){if(v>127)v-=256;return v}{x+=comp(strtonum("0x"$2));y+=comp(strtonum("0x"$3))}$1=="01"{print x,y}' data.txt > data3.txt**
root@kali:~# **tail -n 20 data3.txt**

```
-49 96
-47 98
-47 99
-46 101
-45 101
-43 100
-42 100
-41 100
```

0x02:Forensic&Steg

流量分析

数据提取:Google CTF 2016 Forensic-100

```
tshark -r stego.pcap -T fields -e tcp.urgent_pointer | egrep -vi "^0$"|td
'\n' ','
```

```
root@kali:~/tmp# tshark -r stego.pcap -T fields -e tcp.urgent_pointer | egrep -vi "^0$"|tr '\n' ','
Running as user "root" and group "root". This could be dangerous.
tshark: Lua: Error during loading:
 [string "/usr/share/wireshark/init.lua"]:44: dofile has been disabled due to running Wireshark as superuser. See https://wiki.wireshark.org/Capture
n running Wireshark as an unprivileged user.
67,84,70,123,65,110,100,95,89,111,117,95,84,104,111,117,103,104,116,95,73,116,95,87,97,115,95,73,110,95,84,104,101,95,80,105,99,116,117,114,101,125,
```

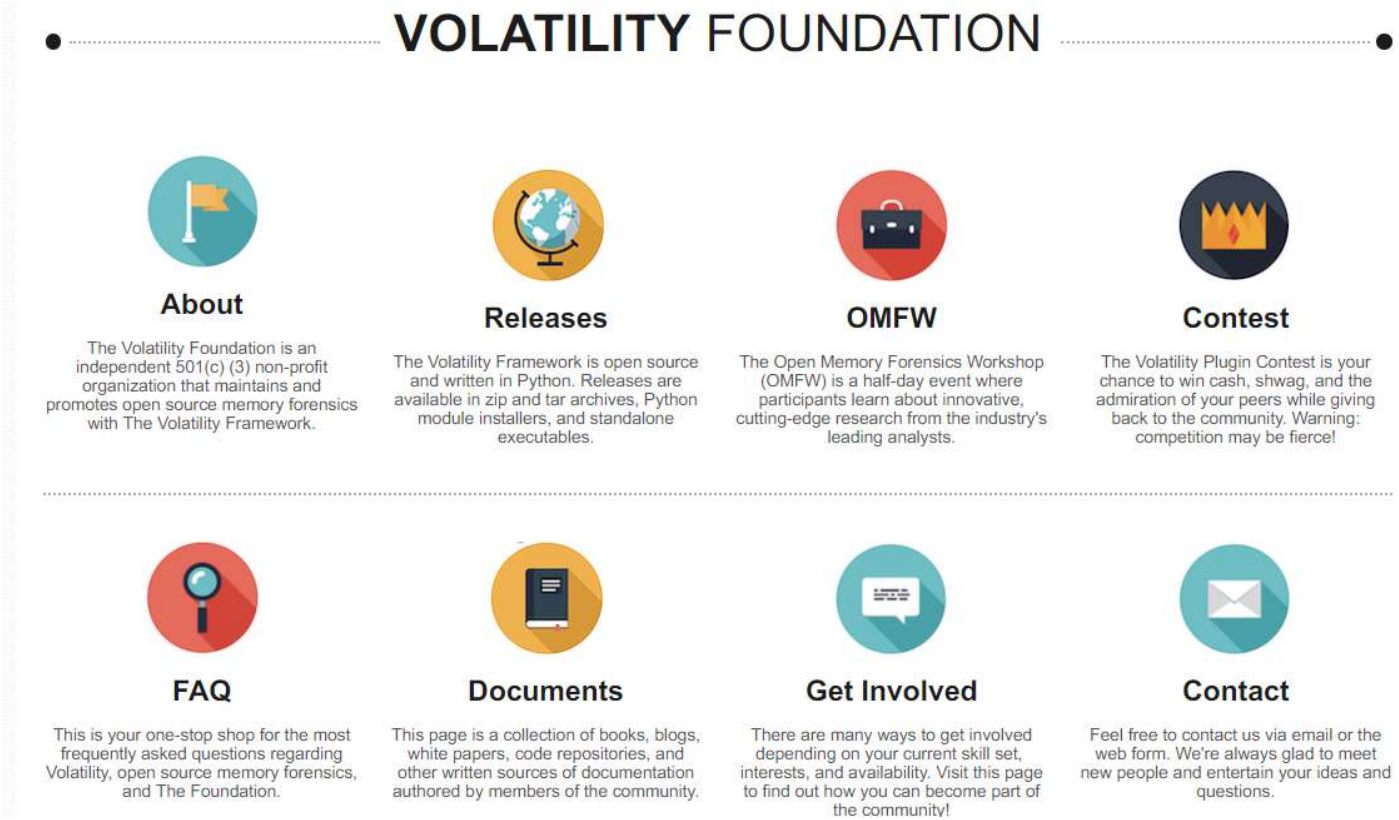
```
root@kali:~# python
Python 2.7.12+ (default, Sep  1 2016, 20:27:38)
[GCC 6.2.0 20160927] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a=[67,84,70,123,65,110,100,95,89,111,117,95,84,104,111,117,103,104,116,95,73
,116,95,87,97,115,95,73,110,95,84,104,101,95,80,105,99,116,117,114,101,125]
>>> print "".join([chr(x) for x in a])
CTF{And_You_Thought_It_Was_In_The_Picture}
>>>
```

0x02:Forensic&Steg

内存文件

Volatility

- 进程
- 文件
- 用户



0x02:Forensic&Steg

内存文件

Volatility

- 确定内存结构
- 查看进程列表
- 根据题目，寻找相关进程，du

演示:
Windows内存取证，Volatilit

1.查看系统相关消息

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  imageinfo
```

2.查看进程

```
volatility -f WIN7.raw --profile=Win7SP1x86 pslist
```

-profile制定内存文件格式,也可不加，pslist/pstree

3.查看内存中的注册列表

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  --profile=WinXPSP3x86  hivelist
```

4.查看注册表中数据

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  --profile=WinXPSP3x86  hivadump -o virtual地址
```

5.查看SAM中账户

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  --profile=WinXPSP3x86  printkey -K "SAM\Domains\Account\Users\Names"
```

6.查看最后登录的用户

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  --profile=WinXPSP3x86  printkey -K "SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon"
```

7.查看正在运行的程序的信息

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  --profile=WinXPSP3x86  userassist
```

8.dump某个具体进程

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  --profile=WinXPSP3x86 memdump  -p 2516  -D /root/tmp
```

9.查看内存中的cmd进程

```
volatility -f DH-CA8822AB9589-20161128-115700.raw  --profile=WinXPSP3x86 cmdscan
```

10.dump系统hash

```
volatility -f WIN7.raw -profile=Win7SP1x86 hashdump -y （注册表 system 的 virtual 地址 ）-s （SAM 的 virtual 地址）
```

0x02:Forensic&Steg

内存文件

Volatility

- 进程

0x82085550	spoolsv.exe	1576	684	13	140	0	0 2016-05-03 04:32:14 UTC+0000
0x81f64560	vmtoolsd.exe	1712	1464	5	145	0	0 2016-05-03 04:32:15 UTC+0000
0x820a3528	ctfmon.exe	1736	1464	1	78	0	0 2016-05-03 04:32:15 UTC+0000
0x81f7d3c0	vmtoolsd.exe	2020	684	7	273	0	0 2016-05-03 04:32:23 UTC+0000
0x8207db28	TPAutoConnSvc.e	512	684	5	99	0	0 2016-05-03 04:32:25 UTC+0000
0x81c26da0	alg.exe	1212	684	6	105	0	0 2016-05-03 04:32:26 UTC+0000
0x81f715c0	wscntfy.exe	1292	1040	1	39	0	0 2016-05-03 04:32:26 UTC+0000
0x81e1f520	TPAutoConnect.e	1972	512	1	72	0	0 2016-05-03 04:32:26 UTC+0000
0x81f9d3e8	TrueCrypt.exe	2012	1464	2	139	0	0 2016-05-03 04:33:36 UTC+0000

0x02:Forensic&Steg

内存文件

Volatility

- 文件

```
root@kali:~/Desktop# volatility  -f forensic_100.raw  filescan | grep host
Volatility Foundation Volatility Framework 2.5
0x000000000201ef90      1      0 R--rw- \Device\HarddiskVolume1\WINDOWS\system
32\svchost.exe
0x00000000020f0268      1      0 R--r-d \Device\HarddiskVolume1\WINDOWS\svchos
t.exe
0x000000000217b748      1      0 R--rw- \Device\HarddiskVolume1\WINDOWS\system
32\drivers\etc\hosts
0x00000000024a7a90      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\system
32\svchost.exe
root@kali:~/Desktop#
```

0x02:Forensic&Steg

内存文件

Volatility

- 用户

```
root@kali:~/quzhen# volatility -f mem.vmem --profile=WinXPSP2x86 printkey -K "SAM\Domains\Account\Users\Names"
Volatility Foundation Volatility Framework 2.6
Legend: (S) = Stable   (V) = Volatile

---------------------------
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
Key name: Names (S)
Last updated: 2016-05-03 03:51:02 UTC+0000

Subkeys:
  (S) Administrator
  (S) Guest
  (S) HelpAssistant
  (S) SUPPORT_388945a0

Values:
REG_DWORD                     : (S) 0
```

目录 content

0x03: Take a look

About Python

Pyc文件

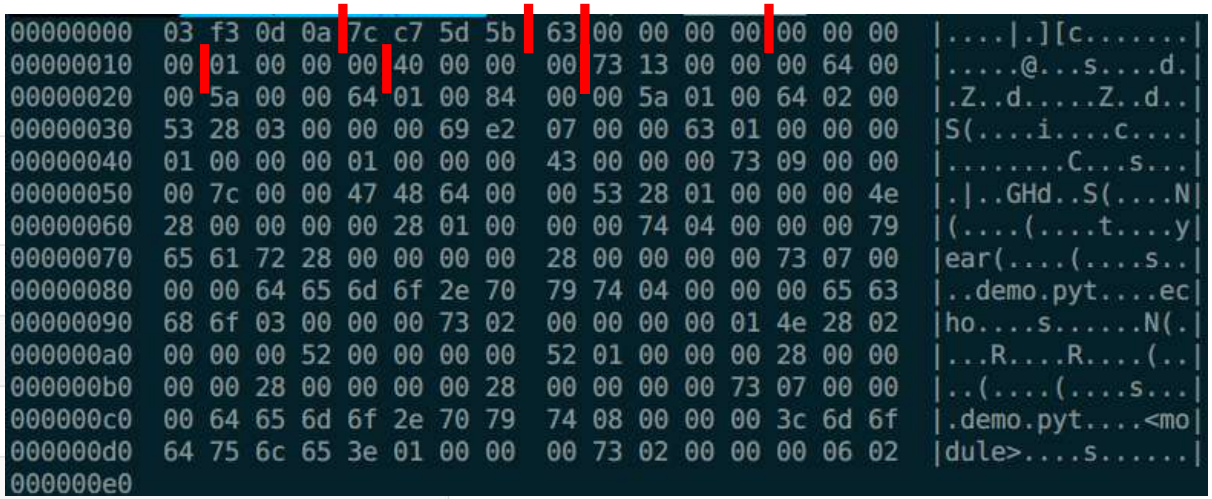
Python 程序编译后得到的字节码文件

Magic [4]	TIMESTAMP [4]	PyCodeObject
表示pyc版本信息	创建时间	

0x03: Take a look

About Python

Pyc文件



First Header	Second Header
co_argcount	0
co_nlocals	1
co_stacksize	1
co_flags	67
co_code	标示0x73，即TYPE_STRING。长度0x0f，即15个字节长度。然后从0x64开始就是co_code内容。

0x03: Take a look

About Python

Pyc文件：co_code 指令序列

- 指令(opcode),分为有参数和无参数两种,以 https://github.com/python/cpython/blob/fc7df0e664198cb05cafd972f190a18ca422989c/Include/opcode.h#L69 划分
- 参数(oparg)

opcode	oparg	opcode	opcode	oparg	...
1 byte	2 bytes	1 byte	1 byte	2 bytes	

0x03: Take a look

About Python

Pyc文件：co_code 指令序列

http://unpyc.sourceforge.net/Opcodes.html

```
>>> import dis

>>> dis.dis(co)
  1           0 LOAD_CONST               0 (2018)
              3 STORE_NAME               0 (year)

  3           6 LOAD_CONST               1 (<code object echo at 0x7f4f00222730, file "demo.py"
, line 3>)
              9 MAKE_FUNCTION            0
             12 STORE_NAME               1 (echo)
             15 LOAD_CONST               2 (None)
             18 RETURN_VALUE
```

```
✘ xu@ubuntu ~/tmp/XMAN/python
✘ xu@ubuntu ~/tmp/XMAN/python cat demo.py
year = 2018

def echo(year):
    print year
xu@ubuntu ~/tmp/XMAN/python
```

0x03: Take a look

About Python

Pyc文件：co_code 指令序列

Stegosaurus¶ **Dead Zone**

原理是在python的字节码文件中,利用冗余空间,将完整的payload代码分散隐藏到这些零零碎碎的空间中.

0x03: Take a look

About Python

Pyc文件：Stegosaurus

```
$ python3 -m stegosaurus ../python_tests/loop.py -s -p"ABCDE" -vv
Read header and bytecode from carrier
BINARY_ADD (0)
BINARY_ADD (0)
BINARY_SUBTRACT (0)
RETURN_VALUE (0)
RETURN_VALUE (0)
Found 5 bytes available for payload
Payload embedded in carrier
BINARY_ADD (65)     <-- A
BINARY_ADD (66)     <-- B
BINARY_SUBTRACT (67) <-- C
RETURN_VALUE (68)   <-- D
RETURN_VALUE (69)   <-- E
```

0x03: Take a look

About Python

Pyc文件：pycdc

A Python Byte-code Disassembler/Decompiler

ERROR??

```
xu in ~/Desktop/mywork/Misc03_350 at 21:20:10 λ /media/xu/Hack/New_Tools/Re/pycdc/pycdc test.pyc
# Source Generated with Decompyle++
# File: test.pyc (Python 2.7)

import sys

def xor(input_data, key):
    result = ''
    for ch in input_data:
        result += chr(ord(ch) ^ key)

    return result


def encrypt(input_data, password):
Unsupported opcode: STOP_CODE
    key = 0
# WARNING: Decompyle incomplete
```

0x03: Take a look

About Python

Pyc文件：pycdc

A Python Byte-code Disassembler/Decompiler

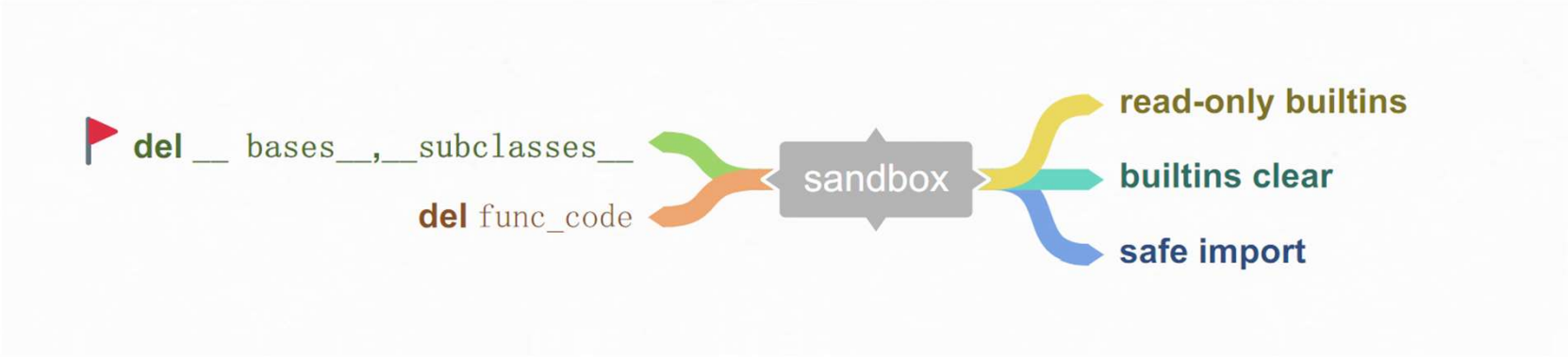
ERROR??

```
xu in ~/Desktop/mywork/Misc03_350 at 21:20:10 λ /media/xu/Hack/New_Tools/Re/pycdc/pycdc test.pyc
# Source Generated with Decompyle++
# File: test.pyc (Python 2.7)

import sys

def xor(input_data, key):
    result = ''
    for ch in input_data:
        result += chr(ord(ch) ^ key)

    return result


def encrypt(input_data, password):
Unsupported opcode: STOP_CODE
    key = 0
# WARNING: Decompyle incomplete
```

0x03: Take a look

About Python

Sandbox Escape



0x03: Take a look

About Python

Sandbox Escape

- Keyword/Import BlackList,WhiteList
  - (__builtins__.__dict__["__import__"])
- Builtins
  - ReadOnly
- subclasses of object
  - [].__class__.__bases__[0].__subclasses__()
- Func_code
  - delete

0x03: Take a look

About Python

The failure of pysandbox

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING !!!
!!!                                                                       !!!
!!! pysandbox is BROKEN BY DESIGN, please move to a new sandboxing         !!!
!!! solution: run python in a sandbox, not the opposite!                   !!!
!!!                                                                       !!!
!!! Learn more about pysandbox failure:                                   !!!
!!! https://lwn.net/Articles/574215/                                      !!!
!!!                                                                       !!!
!!! WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

0x03: Take a look

Word，PDF类：

Word：

- 字体----->隐藏

PDF：

- wbStego4
- pdf-parser
-

0x03: Take a look

PDF:

wbStego4:

ASCII--->二进制--->16进制： 0x20 0x09

1.pdf

编辑为: Hex 运行脚本 运行模板: PDF.bt

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 25 50 44 46 2D 31 2E 35 20 20 20 20 20 09 20 09  %PDF-1.5     . .
0010h: 0D 0A 25 B5 B5 B5 B5 20 20 20 20 20 20 20 20 0D  ..%µµµµ         .
0020h: 0A 31 20 30 20 6F 62 6A 0D 0A 3C 3C 2F 54 79 70  .1 0 obj..<</Typ
0030h: 65 2F 43 61 74 61 6C 6F 67 2F 50 61 67 65 73 20  e/Catalog/Pages
0040h: 32 20 30 20 52 2F 4C 61 6E 67 28 7A 68 2D 43 4E  2 0 R/Lang(zh-CN
0050h: 29 20 2F 53 74 72 75 63 74 54 72 65 65 52 6F 6F  ) /StructTreeRoo
0060h: 74 20 33 35 20 30 20 52 2F 4D 61 72 6B 49 6E 66  t 35 0 R/MarkInf
0070h: 6F 3C 3C 2F 4D 61 72 6B 65 64 20 74 72 75 65 3E  o<</Marked true>
```

20161008103416509320.pdf

编辑为: Hex 运行脚本 运行模板: PDF.bt

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 25 50 44 46 2D 31 2E 35 0D 0A 25 B5 B5 B5 B5 0D  %PDF-1.5..%µµµµ.
0010h: 0A 31 20 30 20 6F 62 6A 0D 0A 3C 3C 2F 54 79 70  .1 0 obj..<</Typ
0020h: 65 2F 43 61 74 61 6C 6F 67 2F 50 61 67 65 73 20  e/Catalog/Pages
0030h: 32 20 30 20 52 2F 4C 61 6E 67 28 7A 68 2D 43 4E  2 0 R/Lang(zh-CN
0040h: 29 20 2F 53 74 72 75 63 74 54 72 65 65 52 6F 6F  ) /StructTreeRoo
0050h: 74 20 33 35 20 30 20 52 2F 4D 61 72 6B 49 6E 66  t 35 0 R/MarkInf
0060h: 6F 3C 3C 2F 4D 61 72 6B 65 64 20 74 72 75 65 3E  o<</Marked true>
0070h: 3E 3E 3E 0D 0A 65 6E 64 6F 62 6A 0D 0A 32 20 30  >>>..endobj..2 0
```

ASCII 码	二进制	十六进制
O	01001111	20 09 20 20 09 09 09 09
b	01100010	20 09 09 20 20 20 09 20
l	01101100	20 09 09 20 09 09 20 20
i =	01101001 =	20 09 09 20 09 20 20 09
v	01110110	20 09 09 09 20 09 09 20
i	01101001	20 09 09 20 09 20 20 09

0x03: Take a look

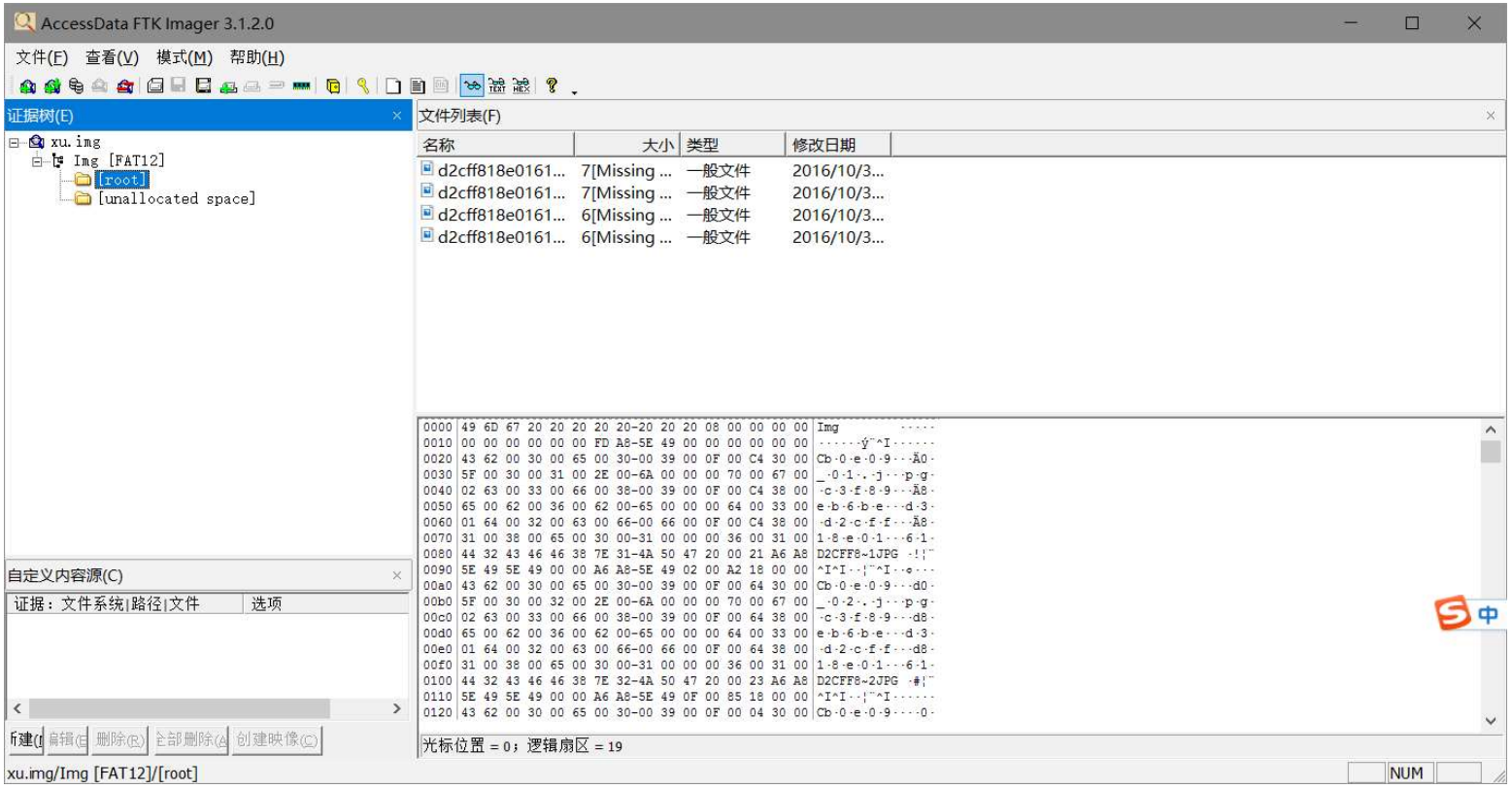
磁盘文件,系统镜像：

1.文件恢复，取证工具：EasyRecovery、FTK,TSK(The Sleuth Kit)
2.常见格式：dd，img，raw
3.磁盘数据：文件系统层、数据层、inode层和文件层
文件系统层：了解磁盘分区的文件系统信息
数据层：包含了文件的真实内容
inode层：数据存储单元与文件属性信息
文件层：文件的具体内容

PS：
磁盘数据与TSK的使用：http://www.freebuf.com/articles/system/57804.html

0x03: Take a look

磁盘文件,系统镜像：



0x03: Take a look

About NTFS数据流

Alternatestreamview：扫描您的NTFS驱动器,可以查找存储在文件系统中所有隐藏的交换数据流.

NTFS交换数据流（alternate data streams，简称ADS）是NTFS磁盘格式的一个特性，在NTFS文件系统下，每个文
件都可以存在多个数据流，就是说除了主文件流之外还可以有许多非主文件流寄宿在主文件流中。它使用资源派生来维
持与文件相关的信息，虽然我们无法看到数据流文件，但是它却是真实存在于我们的系统中的。创建一个数据交换流文
件的方法很简单，命令为"宿主文件:准备与宿主文件关联的数据流文件

0x03: Take a look

About Vedio

AVI，MP4

BCTF2016-catvideo

ffmpeg.exe -i catvideo-497570b7e2811eb52dd75bac9839f19d7bca5ef4.mp4 -r 30.0 fr_%4d.bmp

0x03: Take a look

About Vedio

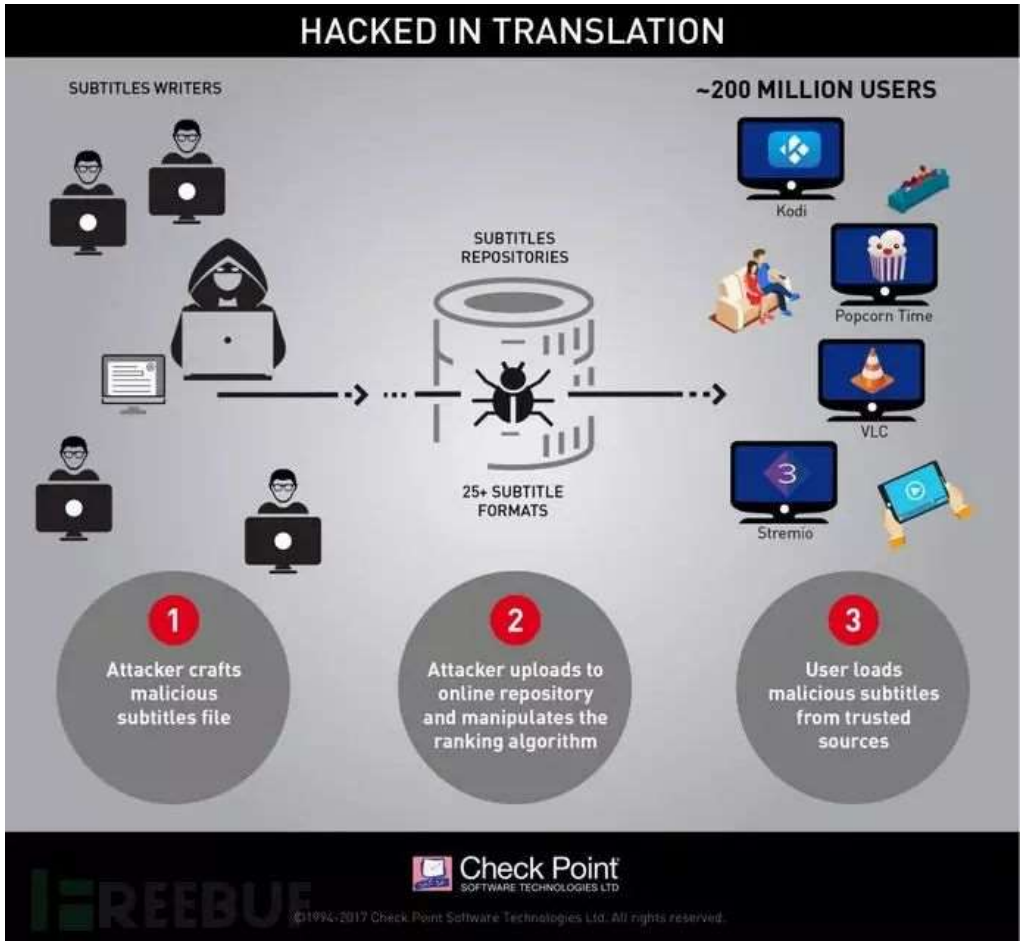
AVI，MP4

Ourstrect

0x03: Take a look

About Vedio

字幕攻击

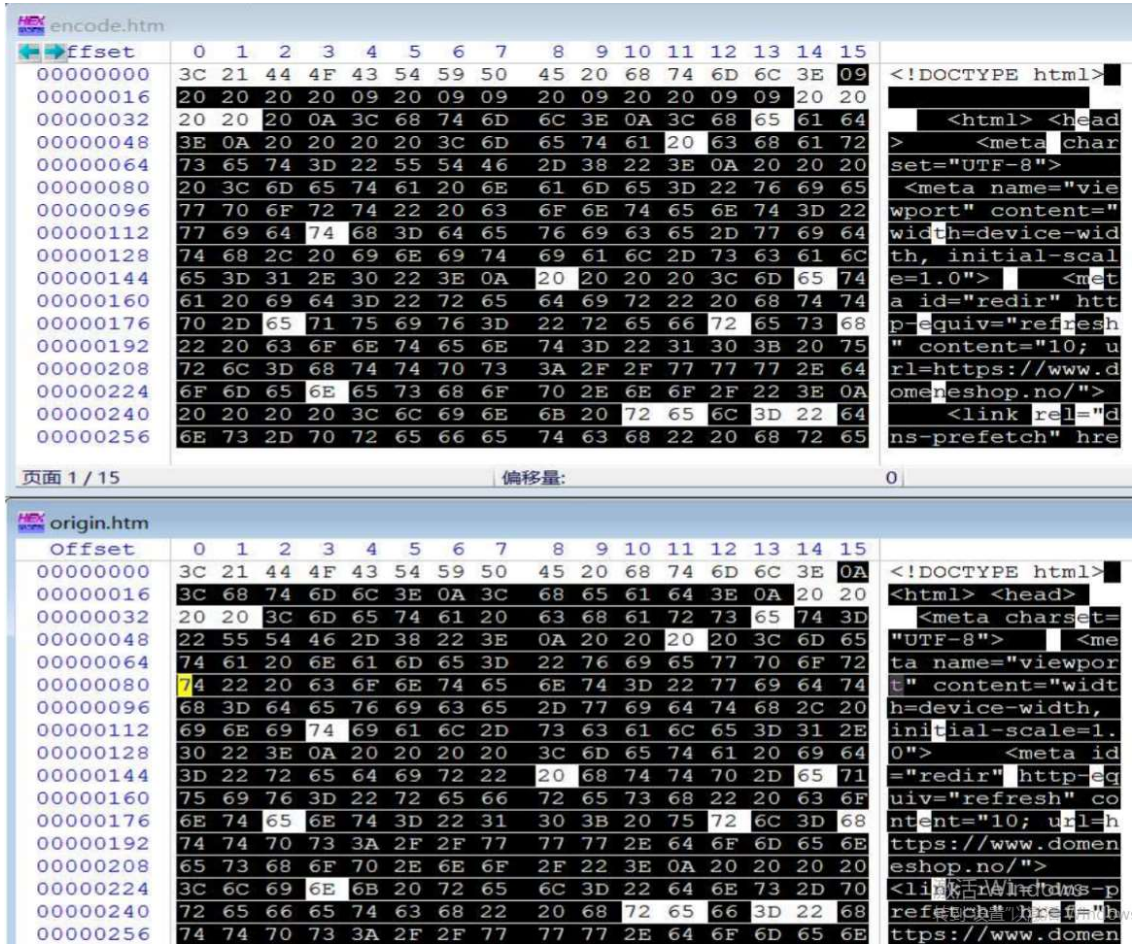


0x03: Take a look

About HTML隐写

Snow.exe：通过插入制表位与空格使得嵌入的数据在浏览器中不可见

http://fog.misty.com/perry/ccs/s



0x03: Take a look

About Game

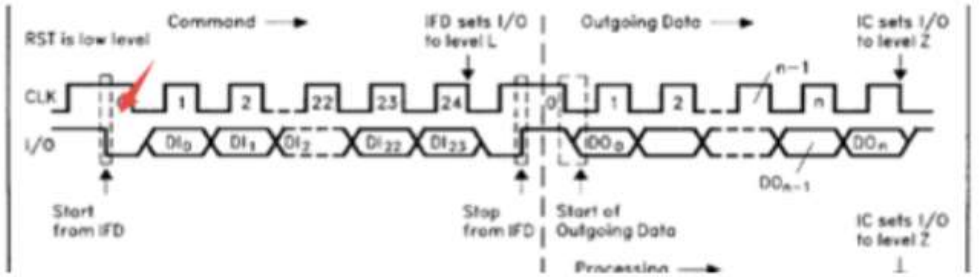
- NES
- Minecraft
- WarCraft III
- CSGO

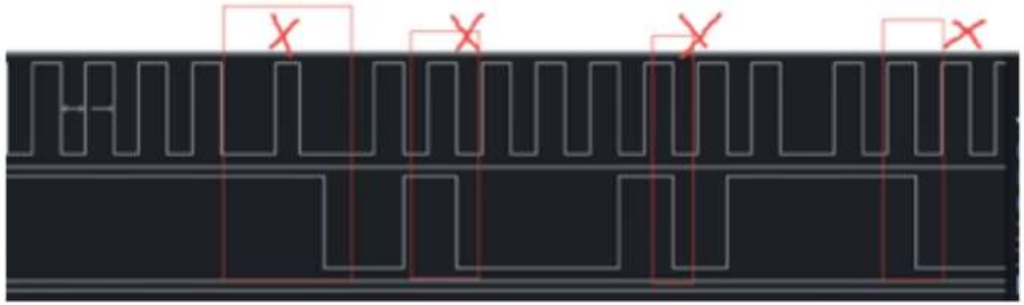
0x03: Take a look

About 工控

- SCTF-2018
  - Modbus

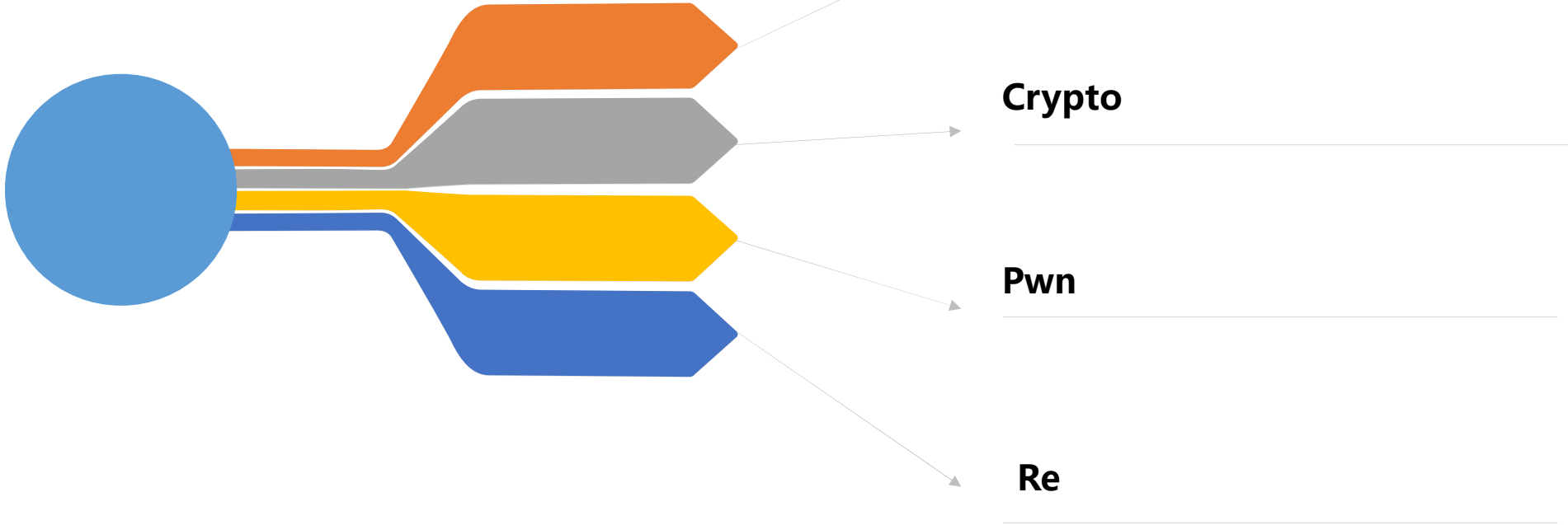
协议





目录 content

0x04: Misc+



0x04: Misc+

+Web

Ciscn-2018

- 验证码破解
  - 机器学习 + 图像处理

SUCTF-2018

- HateIT
  - 文件泄露 + githack

0x04: Misc+

+Crypto

古典密码学

- 栅栏啊
- 凯撒啊
- 维吉尼亚啊
- 字频分析啊
- 培根啊

中低分Misc题

现代密码学

- Aes啊
- Des啊
- Balabala啊
-

高分Misc题

0x04: Misc+

+Pwn

Python SandBox

- Ciscn 2018 Run

```
>>>(lambda r,w:r.seek(0x08de2b8) or w.seek(0x08de8c8) or w.write(r.read(8)) or
().__class__.__bases__[0].__subclasses__()[40]('l'+'s'))
((().__class__.__bases__[0].__subclasses__()[40]('/proc/self/mem','r'),
().__class__.__bases__[0].__subclasses__()[40]('/proc/self/mem', 'w', 0))
```

0x04: Misc+

+Re

- 固件分析
  - 32C3 CTF 2015 config-bin-150
- 病毒分析

Binwalk,winhex, Firmware Modification Kit
Squashfs-tools

After a quick hexdump we might realize that the first 4 bytes are looking pretty much like a magic header.

```
hexdump -C config.bin | head
00000000  43 46 47 31 00 00 32 d8  ef 92 7a b0 5a b6 d8 0d  |CFG1..2...z.Z...|
00000010  30 30 30 30 30 30 00 00  00 05 00 03 00 00 00 00  |000000..........|
```

After searching for "CFG1" with the search engine of our choice, we might find [this](https://heinrichs.io/207) blog entry. This tells us it might be a firmware image of some router (or at least something which looks like it).

There is a windows tool on that website to decrypt the firmware images, lets see if it works for us:

```
decrypting config.bin...
reading config.bin... 13056 bytes read.
parsing header... ok.
setting crypto key... ok.
decrypting... MD5 of plaintext invalid.
```

Hmm, nope. That looks like it might be in deed the right algorithm, but something is wrong (maybe the password).
Well, that would have been too easy otherwise.
So lets try if we can parse the file header with the information provided in that block entry.
As a a python ctypes Structure the header looks like this:

```
class Header(ctypes.BigEndianStructure):
    _fields_ = [
                ("magic", ctypes.ARRAY(ctypes.c_char, 4)),
                ("payload_size", ctypes.c_uint32),
                ("header_md5", ctypes.ARRAY(ctypes.c_ubyte, 8)),
                ("etl", ctypes.ARRAY(ctypes.c_uint8, 7)), # always zero
                ("unused_1", ctypes.c_char),
                ("password_len", ctypes.c_uint16),
                ("padding_len", ctypes.c_uint16),
                ("unused_2", ctypes.ARRAY(ctypes.c_ubyte, 4)),
                ("plaintext_md5", ctypes.ARRAY(ctypes.c_ubyte, 16))
                ]
```

0x04: Misc+

+Pro

Python模拟输入
爆破
算法

问题描述:
name:
serial:78767-77666-76786-87788-77778-66867-66777-86767-66877-77778-88887

根据序列号求name。name就是正确的flag

Hint:
name一共12位,
name[0]='{',name[1]='h',name[2]='d',name[3]='u'
name[5]='b',name[9]='0',name[10]='_',name[11]='}'
其他位的范围为26个小写字母和下划线
暴力循环6min多点（64位i5六代cpu，python）

XCTF

Thanks!