Have fun with Misc

主讲人：Xu

0x00：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：引言

Misc的特点

散发思维

解迷能力

比赛中的Misc

不怀好意的说 现在的Misc基本都是一些套路性的东西 不排除有些是脑洞……

目录 content

01 Recon

02 Encode

03 Forensic&&Stego

04 Challenge Anlysis

目录 content

01 Recon

02 Encode

03 Forensic&&Stego

04 Other

0x01 : Recon

目录 content

01 Recon

02 Encode

03 Forensic&&Stego

04 Other

0x02:Encode

编码 解码以及相互间的转换：

列举的常见编码：

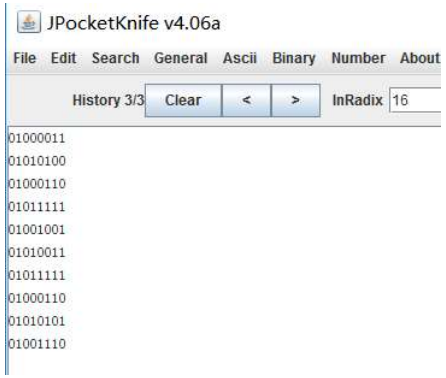
- Bin(二进制）
- Dec(十进制）
- Hex（十六进制）
    - 条形码
    - 二维码
- Base全家桶：
    - Base64 --> Base编码图片
    - Base32
    - Base16
- URL 编码：
- ASCII编码
- 摩尔斯编码
- 曼彻斯特编码/差分曼彻斯特

0x02:Encode

二进制编码的Misc：

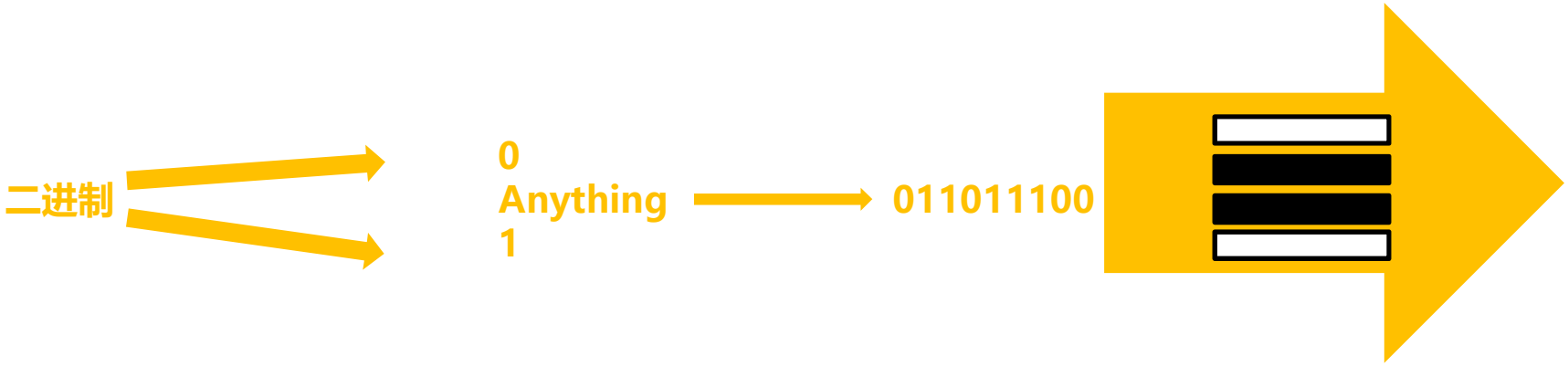
二进制编码是用预先规定的方法将文字、数字或其他对象编成二进制的数码，或将信息、数据转换成规定的二进制电脉冲信号。

（1）二进制编码与字符串之间的转换



0x02:Encode

细谈二进制：



0x02:Encode

二进制编码的Misc：

（1）二进制编码与十进制之间的转换

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

（2）十进制编码与十六进制之间的转换

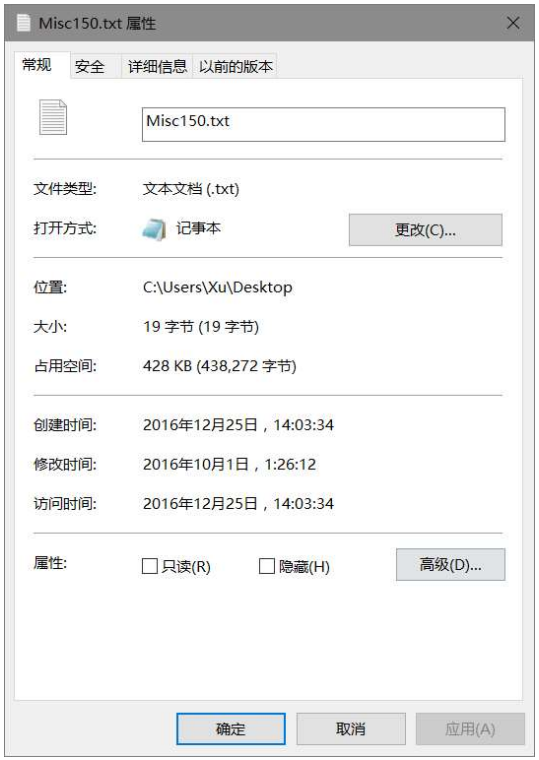
```
>>> hex(100)
'0x64'
>>> int("0x64",16)
100
>>>
```

0x02:Encode

二进制转化：

L-CTF:神奇的压缩文件-150

NTFS数据流隐写+二进制的转化



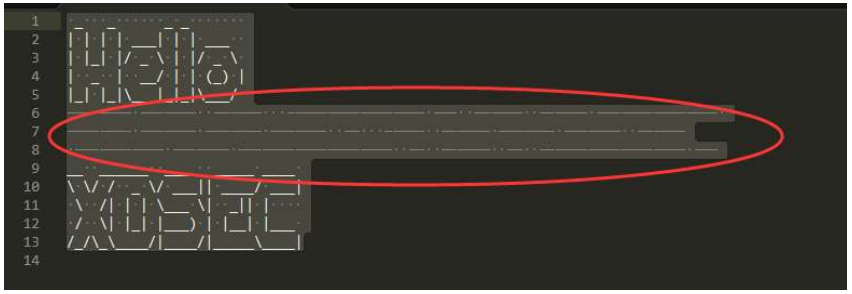
0x02:Encode

XCTF

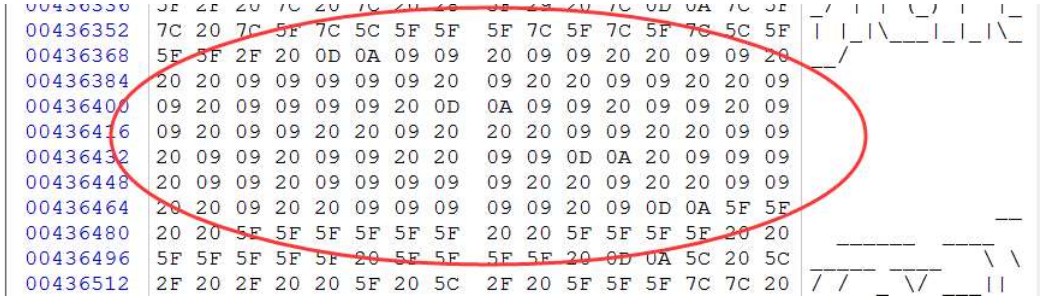
二进制转化：

L-CTF:神奇的压缩文件-150

tab用1替换，空格用0



110110011000111110100110011011110
110110110110010001100110110110011
011101101111100100110010011111101



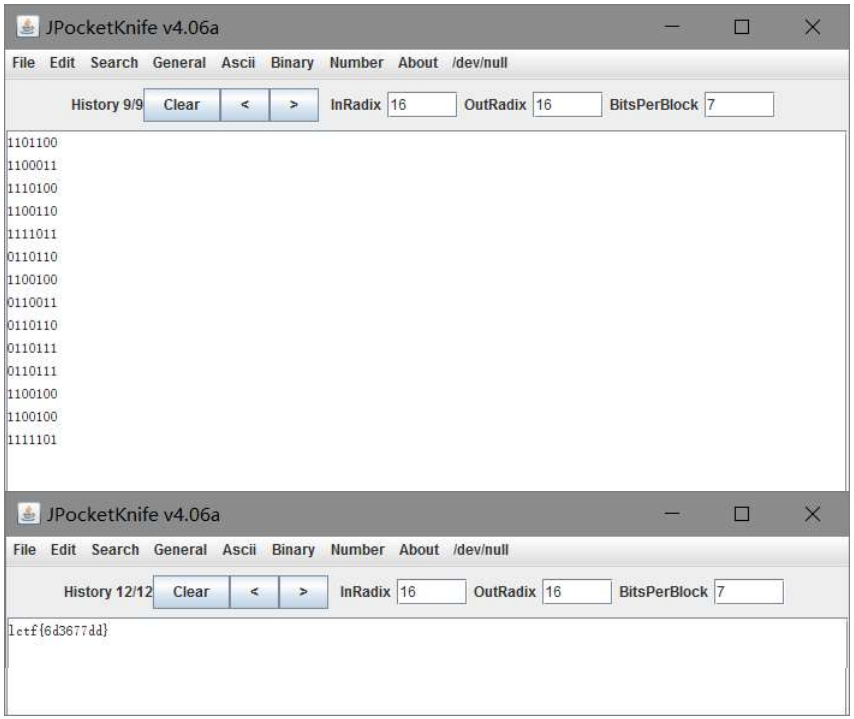
0x02:Encode

二进制转化：

L-CTF:神奇的压缩文件-150

1101100110001111101001100110111 10
1101101101100100011001101101100 11
0111011011111001001100100111110 1

七个一组转化为



0x02:Encode

二进制转化 ---> 二维码：

```
1111111000100001101111111100000101110010110100000110111010100000000010111011011101001000000000101110110111
0111011010010111011000001010101101101000001111111110101010101011111110000000010111011100000000110100110000
0100111011011110101010010000111000000000001010000000010010011010001001110011110111001111000011101111100011
0101000110011100001010100011010001111010110000010100010110000011011101100100001110011100100001011111110100
0000011010100100011110111111101110000110101101110000010000110011000111101011101000110100111110000101110101
0001110100111001011101001001110110110001100000101100011010001100011111110110101101110110111
```

共计625位 ---> 25 * 25

0x02:Encode

二进制转化：

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0110000011001111110011111111100011111100000111000011111111011000001101111111100111111110000011100001100001100
0110000011011000011011000001101100001100011011000000110000110000011011000001101100001100011011000011001100
0110000011011000000011000001101100000000110001100000110000110000011011000001101100001100110001100001111000
0111111111001111110011111111100011111100110000011000011000011000001101111111100110000011011000001100001100
0110000011000000011011000001100000001101111111110000110000110000011011000110001100000110111111110000110000
0110000011011000011011000001101100001101100000110000110000110000011011000110011000001101100000110000110000
0110000011001111110011111111100011111100110000011000011000001111111001100000110111111100110000011000001100
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

共计1177位

```
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0110000011001111110011111111100011111100000111000011111111011000001101111111100111111110000011100001100001100
0110000011011000011011000001101100001100011011000000110000110000011011000001101100001100011011000011001100
0110000011011000000011000001101100000000110001100000110000110000011011000001101100001100110001100001111000
0111111111001111110011111111100011111100110000011000011000011000001101111111100110000011011000001100001100
0110000011000000011011000001100000001101111111110000110000110000011011000110001100000110111111110000110000
0110000011011000011011000001101100001101100000110000110000110000011011000110011000001101100000110000110000
0110000011001111110011111111100011111100110000011000011000001111111001100000110111111100110000011000001100
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
...
```

0x02: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.

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

0x02:Encode

Base64：

base64是怎么解码

- 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的倍数位数二进制
- 删除等于号的个数*8的bit
- 按照6个bit一组转成字符

0x02:Encode

Base64码表：

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		

0x02:Encode

Base64家族之间的转换：

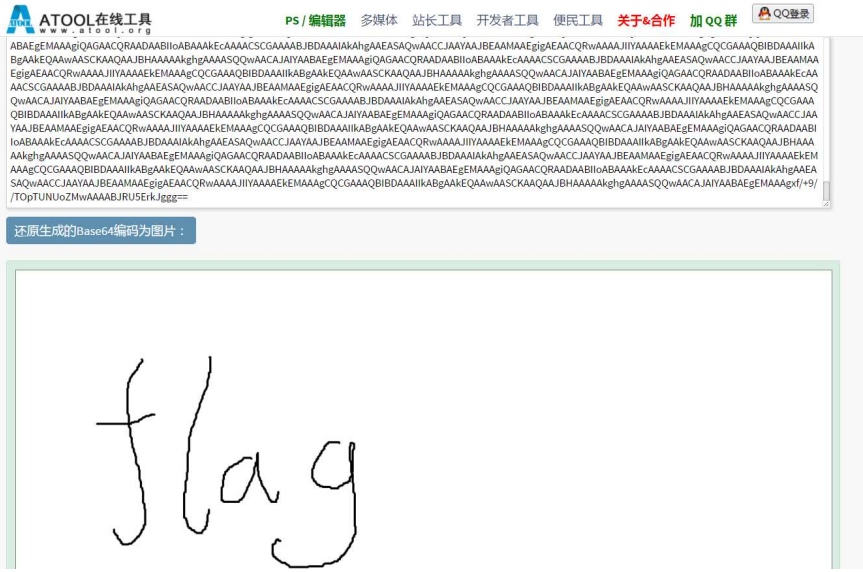
```
>>> str = 'CTF_IS_FUN'
>>> import base64
>>> b64en = base64.b64encode(str)
>>> b64en
'Q1RGX0lTX0ZVTg==='

>>> # 解码
...
>>> b64de = base64.b64decode(b64en)
>>> b64de
'CTF_IS_FUN'
```

0x02:Encode

Base64编码图片：

http://www.atool.org/img2base64.php



0x02:Encode

Base64：

Base64隐写

B								C															
Q						k						M				=							
0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	x	x	x	x	x	x
Q						k						~~M~~				=							

0x02:Encode

Base64的扩展：

Base64隐写

2014 Olympic_ctf

```
U3RlZ2Fub2dyYXBoeSBpcyB0aGUgYXJ0IGFuZCBzY2llbmNlIG9m
IHdyaXRpbmcgaGlkZGVuIG1lc3NhZ2VzIGluIHN1Y2ggYSB3YXkgdGhhdCBubyBvbmV=
LCBhcGFydCBmcm9tIHRoZSBzZW5kZXIgYW5kIGludGVuZGVkIHJlY2lwaWVudCwgc3VzcGU=
Y3RzIHRoZSBleGlzdGVuY2Ugb2YgdGhlIG1lc3M=
YWdlLCBhIGZvcm0gb2Ygc2VjdXJpdHkgdGhyb3VnaCBvYnNjdXJpdHkuIFS=
aGUgd29yZCBzdGVnYW5vZ3JhcGh5IGlzIG9mIEdyZWVrIG9yaWdpbiBhbmQgbWVhbnMgImNvbmNlYW==
bGVkIHdyaXRpbmciIGZyb20gdGhlIEdyZWVrIHdvcmRzIHN0ZWdhbm9zIG1lYW5pbmcgImNv
dmVyZWQgb3IgcHJvdGVjdGVkIiwgYW5kIGdyYXBoZWluIG1lYW5pbmcgInRvIHc=
cml0ZSIuIFRoZSBmaXJzdCByZWNvcmRlZCB1c2Ugb2YgdGhlIHRlcm0gd2FzIGluIDE0OTkgYnkgSm9o
YW5uZXMgVHJpdGhlbWl1cyBpbiBoaXMgU3RlZ2Fub2dyYXBoaWEsIGEgdHJlYV==
dGlzZSBvbiBjcnlwdG9ncmFwaHkgYW5kIHN0ZWdhbm9ncmFwaHkgZGlzZ8==
dWlzZWQgYXMgYSBib29rIG9uIG1hZ2ljLiBHZW5lcmFsbHksIG1lc3P=
YWdlcyB3aWxsIGFwcGVhciB0byBiZSBzb21ldGhpbmcgZWxzZTogaW1hZ2VzLCBhcnRp
Y2xlcywgc2hvcHBpbmcgbGlzdHMsIG9yIHNvbWUgb3R=
aGVyIGNvdmVydGV4dCBhbmQsIGNsYXNzaWNhbGx5LCB0aGUgaGlkZGVuIG1lc3NhZ2UgbWF5IGJlIGluIGludmm=
c2libGUgaW5rIGJldHdlZW4gdGhlIHZpc2libGUgbGluZXMgb2YgYSBwcml2YXRlIGxldHRlci4NCg0KVGhl
IGFkdmFudGFnZSBvZiBzdGVnYW5vZ3JhcGh5LCBvdmVyIGNy
```

https://www.tr0y.wang/2017/06/14/Base64steg/

0x02:Encode

Base64的扩展：

Base64隐写

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

0x02:Encode

摩尔斯编码

文件名：CTF_IS_FUN

Step1：

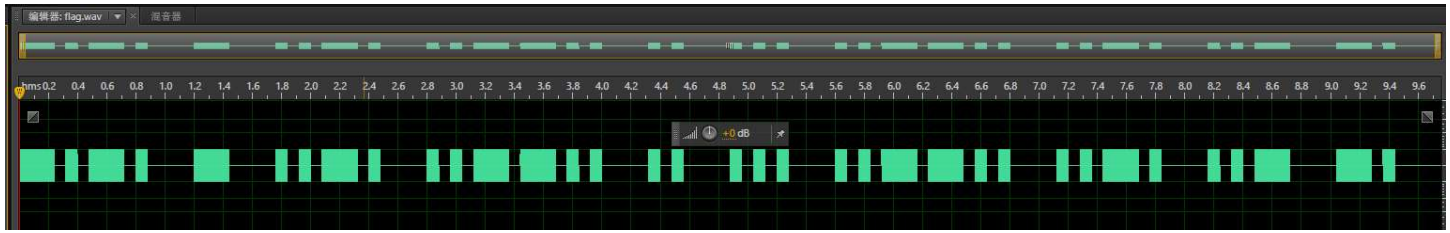
Base64，openssl解码，得到音频文件

```
angr  BinaryNinja-demo.zip  CTF_IS_FUN.wav  CTF-pwn  Other  pycode  secret.data  smap  固件
wings@sw:~/桌面$ openssl base64 -d -in secret.data -out flag
wings@sw:~/桌面$ ls
angr  BinaryNinja-demo.zip  CTF_IS_FUN.wav  CTF-pwn  flag  Other  pycode  secret.data  smap  固件
wings@sw:~/桌面$ file flag
flag: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 8000 Hz
wings@sw:~/桌面$
```

0x02:Encode

摩尔斯编码

Step2：
音频，Morse编码　(au,adobe audition)



-.-. - ..-. ..--.. .. ..--.. ..-. ..- -.

CTF IS FUN

0x02:Encode

图形码：

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

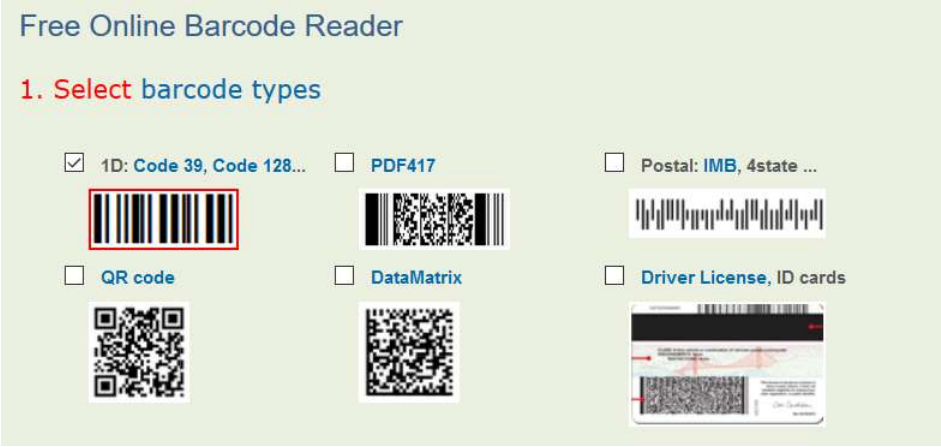
1.条形码：宽度不等的多个黑条和空白，按照一定的编 码规则排列，用以表达一组信息的图形标识符

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 平面（二维方向上）分布的黑白相间的图形 记录数据符号信息

堆叠式/行排式二维条码：Code 16K、Code 49、PDF417

一般CTF赛题中图形码基本包含在这其中



0x02:Encode

CTF中的编码：

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

Quoted-printable 编码

```
#!shell

=E6=95=8F=E6=8D=B7=E7=9A=84=E6=A3=95=E8=89=B2=E7=8B=90=E7=8B=B8=E8=B7=B3=E8
=BF=87=E4=BA=86=E6=87=92=E6=83=B0=E7=9A=84=E7=8B=97
```

0x02:Encode

CTF中的编码：

Brainfuck

```
#!shell
+++++ +++++ [->++ +++++ +++<] >++++ .---. +++++ ++..+ ++.<+ +++++ +++++ [->++ +++++
++++< ]>+++ ++++. <++++ +++[- >---- ---<] >--.< +++++ ++[-> ----- --<]> ----- ----- .<
```

https://www.splitbrain.org/services/ook

0x02:Encode

CTF中的编码：

JSfuck

```
#!shell
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!
+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+
[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+
```

http://www.jsfuck.com/

0x02:Encode

CTF中的编码：

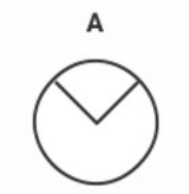
- asp混淆加密
- php混淆加密
- js混淆加密
- VBScript.Encode混淆加密

0x02: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



0x02:Encode

XCTF

工具小结：

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

JPK

CTFCrack

目录 content

01 Recon

02 Encode

03 Forensic&&Stego

04 Other

0x03: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：

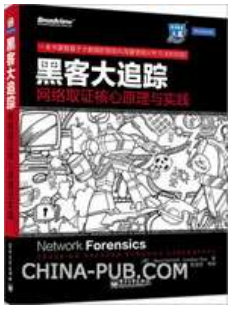
目的一般为发现文件中包含的隐藏字符串(代表需要取证的机密信息)

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



0x03:Forensic&Steg

工具概述：

File：用来鉴定文件类型
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，010：16进制文件编辑器

Exif编辑器

0x03:Forensic&Steg

工具详解：

Binwalk

根据文件头识别包含文件
并进行自动化提取

(提高篇:手工提取)

http://www.freebuf.com/sectool/15266.html

0x03:Forensic&Steg

工具详解：

010 Editor

分析文件16进制
强大的模板功能

编辑为: Hex ▾ 运行脚本 ▾ 运行模板: JPG.bt ▾

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    0123456789ABCDEF
0000h:  FF D8 FF E0 00 10 4A 46 49 46 00 01 01 01 00 48    ÿØÿà..JFIF.....H
0010h:  00 48 00 00 FF DB 00 43 00 01 01 01 01 01 01 01    .H..ÿÛ.C........
0020h:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    ................
0030h:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    ................
0040h:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    ................
0050h:  01 01 01 01 01 01 01 01 01 FF DB 00 43 01 01 01    .........ÿÛ.C...
0060h:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    ................
0070h:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    ................
0080h:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01    ................
0090h:  01 01 01 01 01 01 01 01 01 01 01 01 01 01 FF C0    ..............ÿÀ
00A0h:  00 11 08 02 46 04 00 03 01 11 00 02 11 01 03 11    ....F...........
00B0h:  01 FF C4 00 1F 00 00 01 04 03 01 01 01 01 00 00    .ÿÄ............
00C0h:  00 00 00 00 00 00 05 03 04 06 07 02 08 09 00 01    ................
00D0h:  0A 0B FF C4 00 6D 10 00 00 04 02 05 07 08 06 07    ..ÿÄ.m..........
00E0h:  05 05 04 06 04 04 17 01 02 03 04 05 11 00 06 21    ...............!
00F0h:  31 41 07 12 13 14 51 61 71 08 15 22 24 81 91 A1    1A....Qaq.."$.'¡
0100h:  F0 23 25 32 B1 C1 D1 09 33 34 35 44 E1 F1 16 42    ð#%2±ÁÑ.345Dáñ.B
0110h:  43 45 54 52 53 55 62 64 17 63 65 74 0A 26 36 73    CETRSUbd.cet.&6s
0120h:  75 84 72 83 85 94 18 27 46 56 82 92 93 95 A2 A3    u„rƒ…”.'FV,'"•¢£
0130h:  A4 A5 B4 66 C4 C2 D2 E2 19 47 B2 B3 B5 D4 E4 E5    ¤¥´fÄÂÒâ.G²³µÔäå
```

模板的结果 - JPG.bt

名称	值	开始	大小	颜色	注释
struct JPGFILE jpgfile		0h	4D59Bh	Fg: Bg:	
enum M_ID SOIMarker	M_SOI (FFD8h)	0h	2h	Fg: Bg:	
struct APP0 app0		2h	12h	Fg: Bg:	
struct DQT dqt[0]		14h	45h	Fg: Bg:	
struct DQT dqt[1]		59h	45h	Fg: Bg:	
struct SOFx sof0		9Eh	13h	Fg: Bg:	
struct DHT dht[0]		B1h	21h	Fg: Bg:	
struct DHT dht[1]		D2h	6Fh	Fg: Bg:	
struct DHT dht[2]		141h	1Eh	Fg: Bg:	
struct DHT dht[3]		15Fh	60h	Fg: Bg:	
struct SOS scanStart		1BFh	Bh	Fg: Bg:	
char scanData[316367]		1CAh	4D3CFh	Fg: Bg:	
enum M_ID EOIMarker	M_EOI (FFD9h)	4D599h	2h	Fg: Bg:	

0x03:Forensic&Steg

常见文件特征：

熟悉：

1.JPEG、PNG、GIF、BMP等图片文件
2.ZIP、 RAR、TAR、7z等压缩文件
3.mp3、wma、avi等音 视频文件

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

jpg jpeg	JPEG raw or in the JFIF or Exif file format	0	ÿØÿÛ	FF D8 FF DB
			ÿØÿà ..J F IF..	FF D8 FF E0 nn nn 4A 46 49 46 00 01
			ÿØÿá ..E x if..	FF D8 FF E1 nn nn 45 78 69 66 00 00

0x03:Forensic&Steg

常见文件特征：

gif	Image file encoded in the Graphics Interchange Format (GIF)[2]	0	GIF87a GIF89a	47 49 46 38 37 61 47 49 46 38 39 61
png	Image encoded in the Portable Network Graphics format[8]	0	.PNG....	89 50 4E 47 0D 0A 1A 0A
bmp dib	BMP file, a bitmap format used mostly in the Windows world	0	BM	42 4D

0x03:Forensic&Steg

常见文件特征：

zip				50 4B 03 04
rar	RAR archive version 1.50 onwards[6]	0	Rar!...	52 61 72 21 1A 07 00
tar	tar archive[28]	0x101	ustar.00	75 73 74 61 72 00 30 30
			ustar .	75 73 74 61 72 20 20 00

0x03: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

0x03: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

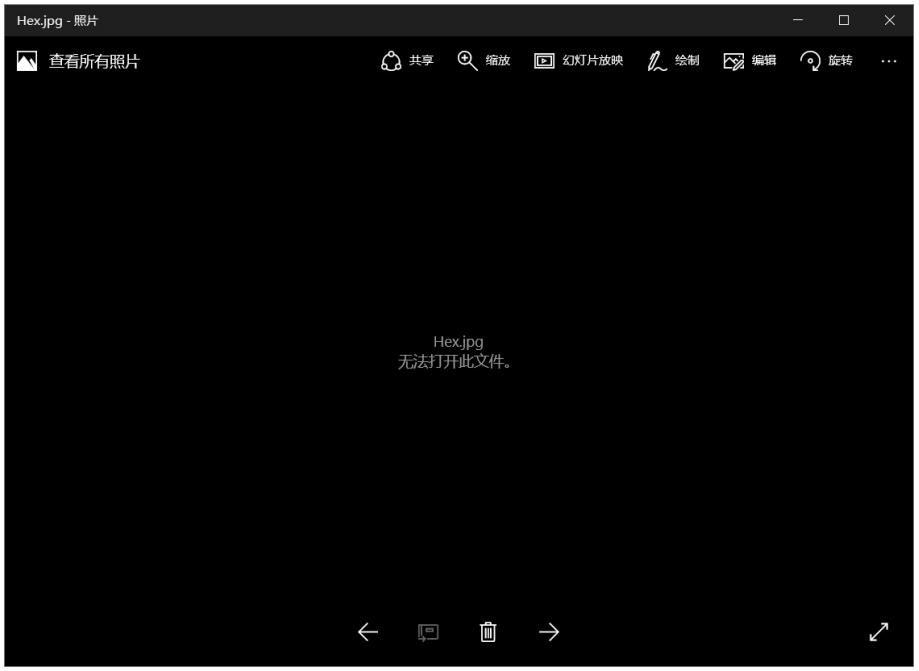
MagicEXIF

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

0x03:Forensic&Steg

损坏的JPG



0x03:Forensic&Steg

损坏的JPG

JPG文件.jpg

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	FF	D9	FF	E0	00	10	4A	46	49	46	00	01	01	00	00	01	ÿÙÿà JFIF
00000010	00	01	00	00	FF	DB	00	43	00	14	0E	0F	12	0F	0D	14	ÿÛ C
00000020	12	10	12	17	15	14	18	1E	32	21	1E	1C	1C	1E	3D	2C	2! =,
00000030	2E	24	32	49	40	4C	4B	47	40	46	45	50	5A	73	62	50	.$2I@LKG@FEPZsbP
00000040	55	6D	56	45	46	64	88	65	6D	77	7B	81	82	81	4E	60	UmVEFdˆemw{ , N`
00000050	8D	97	8C	7D	96	73	7E	81	7C	FF	DB	00	43	01	15	17	—Œ}–s~ |ÿÛ C
00000060	17	1E	1A	1E	3B	21	21	3B	7C	53	46	53	7C	7C	7C	7C	;!!;|SFS||||
00000070	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	||||||||||||||||
00000080	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	||||||||||||||||
00000090	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	7C	FF	C0	||||||||||||||ÿÀ
000000A0	00	11	08	02	80	02	80	03	01	22	00	02	11	01	03	11	€ € "

FF D9 → FF D8

0x03:Forensic&Steg

深入理解 JPEG 图像格式 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。

0x03: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

标准数据块：

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

0x03: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

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)

0x03:Forensic&Steg

PNG文件格式：

标准数据块-调色板数据块PLTE：索引彩色图像有关

需要在IDAT块之前

域的名称	字节数	说明
Red	1 byte	0 = 黑，255 = 红
Green	"> 0 = 黑，255 = 绿	
Blue	1 byte	0 = 黑，255 = 蓝

0x03:Forensic&Steg

PNG文件格式：

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

0x03: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)

0x03:Forensic&Steg

PNG文件格式：

数据块类别：

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

0x03:Forensic&Steg

PNG文件格式：

mma-ctf-2nd-2016 ninth-100

IDAT 隐藏数据：

PNG图片中可以存在多个IDAT块 因此这也成了一个隐写的途径

```
import png
import zlib
import string

def get_decompressed_data(filename):
        img = png.Reader(filename)
        for chunk in img.chunks():
                if chunk[0] == "IDAT":
                        print len(chunk[1])
                        return zlib.decompress(chunk[1])
```

0x03:Forensic&Steg

PNG文件格式：

mma-ctf-2nd-2016 ninth-100

I通过计算 得知总的IDAT块大小为4071274

```
data = get_decompressed_data("ninth.png")
len_data = str(len(data))
print '{0}'.format('len_data = :' + len_data)
```

然而有效的IDAT块大小却只有3053648

```
image_width = 1200
image_height = 848
bytes_per_pixel = 3
bytes_per_filter = 1
data = data[(image_width*bytes_per_pixel + bytes_per_filter)*image_height:]
```

0x03:Forensic&Steg

PNG文件格式：

mma-ctf-2nd-2016 ninth-100

其余对PNG图片而言都是多余数据..我们将可以打印字符打印出来

```
import png
import zlib
import string

def get_decompressed_data(filename):
        img = png.Reader(filename)
        for chunk in img.chunks():
                if chunk[0] == "IDAT":
                        #print len(chunk[1])
                        return zlib.decompress(chunk[1])

data = get_decompressed_data("ninth.png")
len_data = str(len(data))
print '{0}'.format('len_data = :' + len_data)

image_width = 1200
image_height = 848
bytes_per_pixel = 3
bytes_per_filter = 1
data = data[(image_width*bytes_per_pixel + bytes_per_filter)*image_height:]

print ''.join([x for x in data if x in string.printable])
```

0x03:Forensic&Steg

PNG文件格式：

ssctf　利用pngcheck 对图片进行检查

```
λ pngcheck.exe -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).
```

0x03:Forensic&Steg

PNG文件格式：

ssctf　利用pngcheck 对图片进行检查

可以看到，正常的块的length是在65524的时候就满了，而倒数第二个IDAT块长度是45027，最后一个长度是138，很明显最后一个IDAT块是有问题的，因为他本来应该并入到倒数第二个未满的块里

```
1421280  95 00 FA 54 0D 21 BD BA  02 FF 3F 01 E7 98 5E 68  • úT !½º ÿ? ç˜^h
1421296  95 8F CD 00 00 00 8A 49  44 41 54 78 9C 5D 91 01  • Í   ŠIDATxœ]'
1421312  12 80 40 08 02 BF 04 FF  FF 5C 75 29 4B 55 37 73   €@  ¿ ÿÿ\u)KU7s
1421328  8A 21 A2 7D 1E 49 CF D1  7D B3 93 7A 92 E7 E6 03  Š!¢} IÏÑ}³"z'çæ
1421344  88 0A 6D 48 51 00 90 1F  B0 41 01 53 35 0D E8 31  ˆ mHQ   °A S5 è1
1421360  12 EA 2D 51 C5 4C E2 E5  85 B1 5A 2F C7 8E 88 72   ê-QÅLâå…±Z/ÇŽˆr
1421376  F5 1C 6F C1 88 18 82 F9  3D 37 2D EF 78 E6 65 B0  õ oÁˆ ‚ù=7-ïxæe°
1421392  C3 6C 52 96 22 A0 A4 55  88 13 88 33 A1 70 A2 07  ÃlR–" ¤Uˆ ˆ3¡p¢
1421408  1D DC D1 82 19 DB 8C 0D  46 5D 8B 69 89 71 96 45   ÜÑ‚ ÛŒ F]‹i‰q–E
1421424  ED 9C 11 C3 6A E3 AB DA  EF CF C0 AC F0 23 E7 7C  íœ Ãjã«ÚïÏÀ¬ð#ç|
1421440  17 C7 89 76 67 D9 CF A5  A8 00 00 00 00 49 45 4E   Ç‰vgÙÏ¥¨    IEN
1421456  44 AE 42 60 82                                    D®B`‚
```

PNG的IDAT是使用从LZ77派生的无损数据压缩算法，可以用zlib解压。

0x03:Forensic&Steg

PNG文件格式：

ssctf　利用pngcheck 对图片进行检查

得到16进制的以方便python处理，前面的4字节是长度 然后是标志位IDAT 然后开始是数据，直到 D9 CF A5 A8是crc32校验位。 所以实际的数据是：

789C5D91011280400802BF04FFFF5C75294B5537738A21A27D1E49CFD17DB3937A92E7
E603880A6D485100901FB0410153350DE83112EA2D51C54CE2E585B15A2FC78E8872F51
C6FC1881882F93D372DEF78E665B0C36C529622A0A45588138833A170A2071DDCD1821
9DB8C0D465D8B6989719645ED9C11C36AE3ABDAEFCFC0ACF023E77C17C7897667

0x03:Forensic&Steg

PNG文件格式：

例题2： ssctf　利用pngcheck 对图片进行检查

发现解出来了一些3031的字符串，30和31是hex的 0和1的编码，再解一次hex得到一串625长度的01字符串。

```
#!/usr/bin/env python
import Image
MAX = 25
pic = Image.new("RGB",(MAX, MAX))
str = "1111111000100001101111111100000101110010110100000110111010100000000010111011011101001000000000101110110111101
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")
```

0x03:Forensic&Steg

PNG文件格式：

MoreHight　修改PNG图片的高度　隐藏数据

Name	Value	Start
struct PNG_SIGNATURE sig		0h
struct PNG_CHUNK chunk[0]	IHDR (Critical, Pu...	8h
uint32 length	13	8h
union CTYPE type	IHDR	Ch
struct PNG_CHUNK_IHDR ihdr	528 x 700 (x8)	10h
uint32 width	528	10h
uint32 height	700	14h
ubyte bits	8	18h
enum PNG_COLOR_SPACE_TYPE color_type	AlphaTrueColor (6)	19h
enum PNG_COMPR_METHOD compr_method	Deflate (0)	1Ah
enum PNG_FILTER_METHOD filter_method	AdaptiveFiltering (0)	1Bh
enum PNG_INTERLACE_METHOD interlace_method	NoInterlace (0)	1Ch
uint32 crc	A3F8BDBBh	1Dh

0x03:Forensic&Steg

PNG文件格式：

MoreHight　修改PNG图片的高度 隐藏数据

Name	Value	Start	
struct PNG_SIGNATURE sig		0h	8h
struct PNG_CHUNK chunk[0]	IHDR (Critical, Pu...	8h	19h
uint32 length	13	8h	4h
union CTYPE type	IHDR	Ch	4h
struct PNG_CHUNK_IHDR ihdr	528 x 800 (x8)	10h	Dh
uint32 width	528	10h	4h
uint32 height	800	14h	4h
ubyte bits	8	18h	1h
enum PNG_COLOR_SPACE_TYPE color_type	AlphaTrueColor (6)	19h	1h
enum PNG_COMPR_METHOD compr_method	Deflate (0)	1Ah	1h
enum PNG_FILTER_METHOD filter_method	AdaptiveFiltering (0)	1Bh	1h
enum PNG_INTERLACE_METHOD interlace_method	NoInterlace (0)	1Ch	1h
uint32 crc	A3F8BDBBh	1Dh	4h

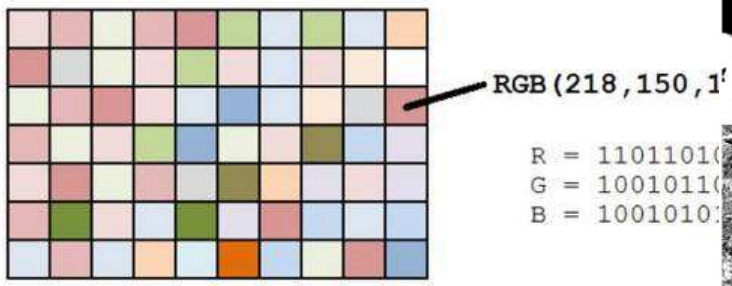
0x03:Forensic&Steg

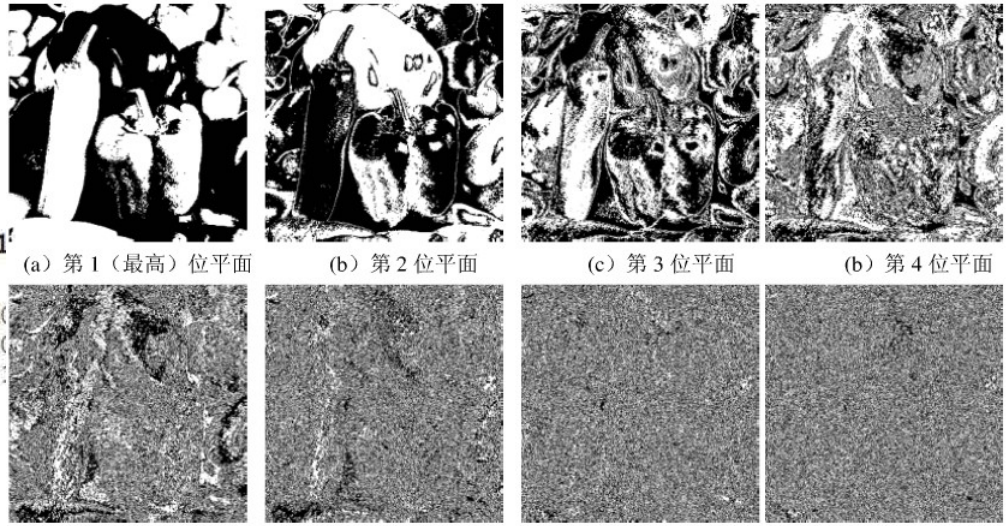
LSB隐写

Stegsolve:图片通道查看器，图片隐写必备

LSB隐写介绍：

最低有效位 (Least Significant Bit)





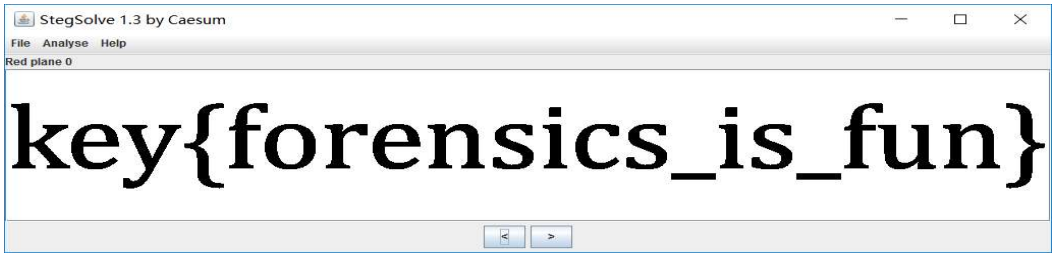
(a）第 1（最高）位平面 (b）第 2 位平面 (c）第 3 位平面 (b）第 4 位平面

(e）第 5 位平面 (f）第 6 位平面 (g）第 7 位平面 (h）第 8（最低）位平面

0x03:Forensic&Steg

LSB隐写

chal.png



此类LSB仅适用于PNG与BMP格式的图片文件
因为PNG属于无损压缩，BMP属于无压缩
而JPG对像素做了有损压缩
我们所修改隐藏的信息会被压缩的过程破坏

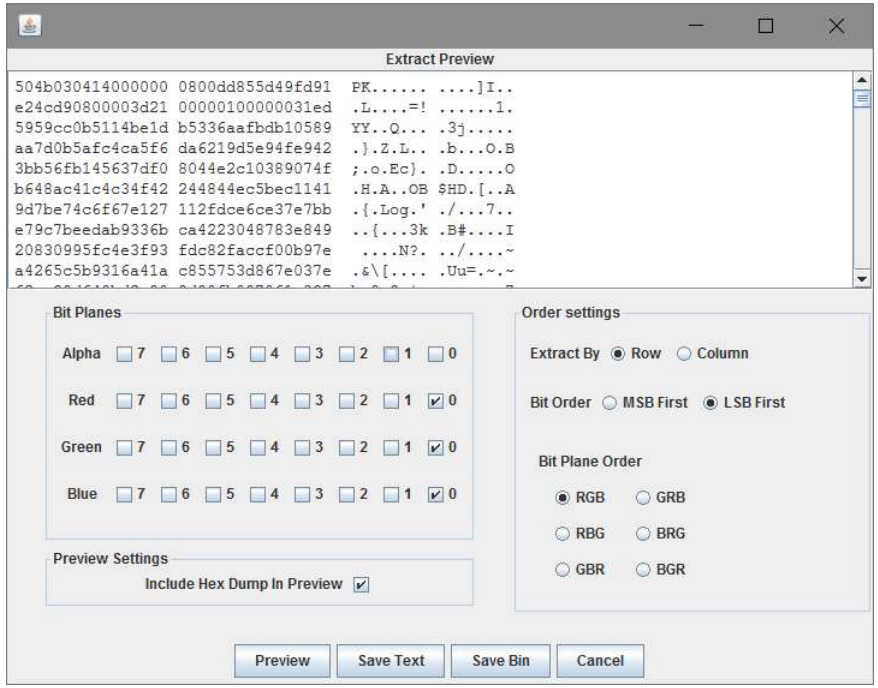
0x03:Forensic&Steg

StegSolve：

HCTF-2016-Misc

0x03:Forensic&Steg

StegSolve：




```
root@kali:~/tmp# strings 12|grep hctf
hctf{dd0gf4c3tok3yb0ard4g41n~~~}
root@kali:~/tmp#
```

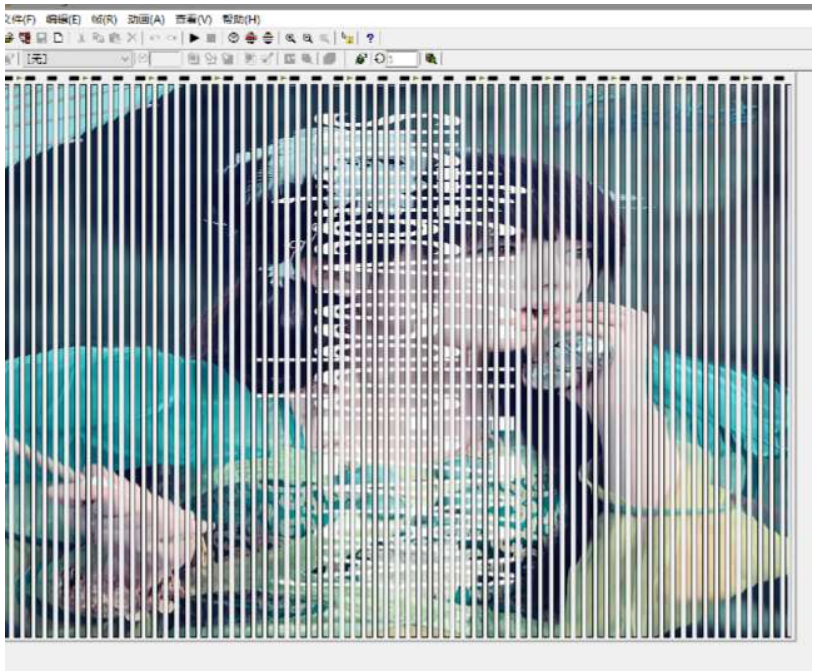
0x03:Forensic&Steg

GIF:

文件头

空间

时间



0x03:Forensic&Steg

XCTF

音频文件：

Adobe audition/Audacity：频谱，波形的处理

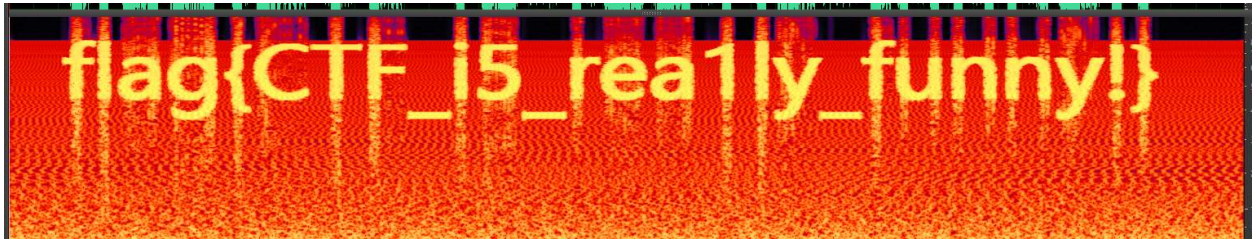
- 频谱
- 波形
- 隐写软件

0x03:Forensic&Steg

XCTF

音频文件：

- 频谱
- 波形
- 隐写软件

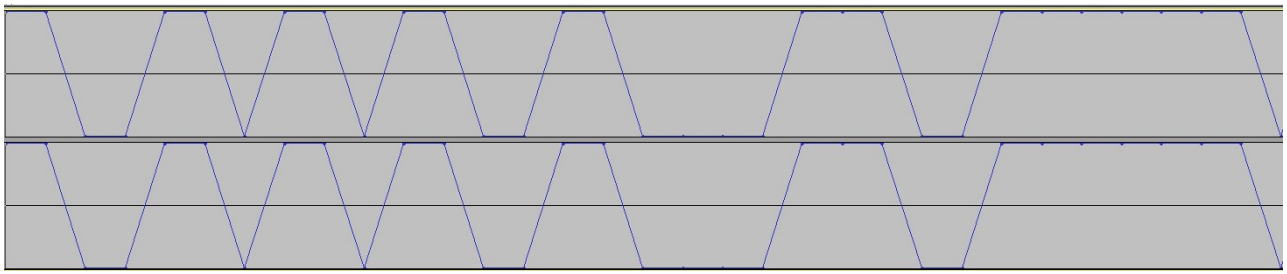


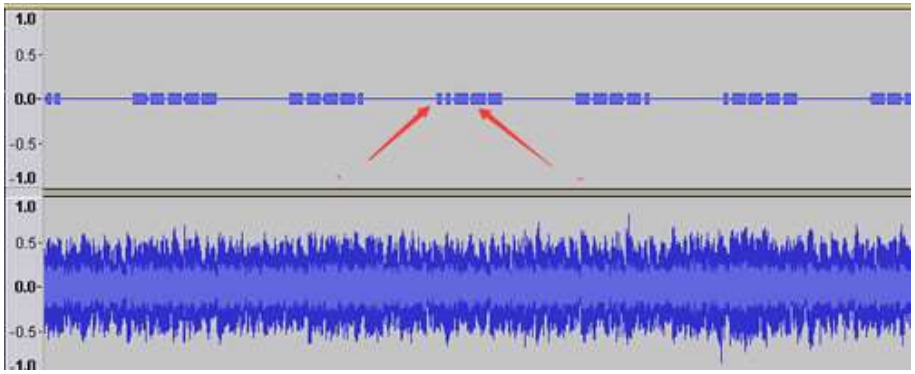
0x03:Forensic&Steg

XCTF

音频文件：

- 频谱
- **波形**
- 隐写软件





0x03:Forensic&Steg

音频文件：

- 频谱
- 波形
- **隐写软件**
  - **MP3Stego** Decode.exe -X hong.mp3
  - **Slienteye**

0x03:Forensic&Steg

ZIP文件格式：

组成：

-压缩源文件数据区

- 文件头
- 文件数据
- （数据描述符）

-核心目录

-目录结束标识

user.zip

编辑为: Hex 运行脚本 运行模板: ZIP.bt

```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 50 4B 03 04 14 00 00 00 08 00 E2 A6 9B 49 72 7D   PK....|..|..â¦›Ir}
0010h: DB 2D 0E 00 00 00 0C 00 00 00 08 00 00 00 75 73   Û-............us
0020h: 65 72 2E 74 78 74 4B CB 49 4C AF 3E B7 71 CF E5   er.txtKËIL¯>·qÏå
0030h: C3 77 6A 01 50 4B 01 02 1F 00 14 00 00 00 08 00   Ãwj.PK..........
0040h: E2 A6 9B 49 72 7D DB 2D 0E 00 00 00 0C 00 00 00   â¦›Ir}Û-........
0050h: 08 00 24 00 00 00 00 00 00 00 20 00 00 00 00 00   ..$....... .....
0060h: 00 00 75 73 65 72 2E 74 78 74 0A 00 20 00 00 00   ..user.txt.. ...
0070h: 00 00 01 00 18 00 CE 86 B9 71 40 60 D2 01 2F B3   ......Î†¹q@`Ò./³
0080h: EF 18 70 5F D2 01 2F B3 EF 18 70 5F D2 01 50 4B   ï.p_Ò./³ï.p_Ò.PK
0090h: 05 06 00 00 00 00 01 00 01 00 5A 00 00 00 34 00   ..........Z...4.
00A0h: 00 00 00 00                                       ....
```

模板的结果 - ZIP.bt

名称	值	开始	大小	颜色
struct ZIPFILERECORD record	user.txt	0h	34h	Fg: Bg:
char frSignature[4]	PK	0h	4h	Fg: Bg:
ushort frVersion	20	4h	2h	Fg: Bg:
ushort frFlags	0	6h	2h	Fg: Bg:
enum COMPTYPE frCompres…	COMP_DEFLATE (8)	8h	2h	Fg: Bg:
DOSTIME frFileTime	20:55:04	Ah	2h	Fg: Bg:
DOSDATE frFileDate	12/27/2016	Ch	2h	Fg: Bg:
uint frCrc	2DDE7D72h	Eh	4h	Fg: Bg:
uint frCompressedSize	14	12h	4h	Fg: Bg:
uint frUncompressedSize	12	16h	4h	Fg: Bg:
ushort frFileNameLength	8	1Ah	2h	Fg: Bg:
ushort frExtraFieldLength	0	1Ch	2h	Fg: Bg:
char frFileName[8]	user.txt	1Eh	8h	Fg: Bg:
uchar frData[14]		26h	Eh	Fg: Bg:
struct ZIPDIRENTRY dirEntry	user.txt	34h	5Ah	Fg: Bg:
struct ZIPENDLOCATOR endL…		8Eh	16h	Fg: Bg:

0x03:Forensic&Steg

ZIP文件格式：

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
6	2	General purpose bit flag	通用比特标志位(置比特0位=加密，详情见后)
8	2	compression method	压缩方式
10	4	compressed size	压缩后的大小
22	4	Uncompressed size	未压缩的大小
26	4	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

压缩源文件数据区:

- 文件头
- 文件数据
- （数据描述符）

Offset	Bytes	Description	译
0	4	crc-32	CRC-32校验码
4	4	compressed size	压缩后的大小
8	4	uncompressed size	未压缩的大小

0x03:Forensic&Steg

ZIP文件格式：

核心目录:

Offset	Bytes	Description	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校验码
20	4	Compressed size	压缩后的大小
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 header	本地文件头的相对位移
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

0x03:Forensic&Steg

ZIP文件格式：

目录结束标识:

Offset	Bytes	Description	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	number of the disk with the start of the central directory	核心目录开始位置的磁盘编号
8	2	total number of entries in the central directory on this disk	该磁盘上所记录的核心目录数量
10	2	total number of entries in the central directory	核心目录结构总数
12	2	Size of central directory (bytes)	核心目录的大小
16	4	offset of start of central directory with respect to the starting disk number	核心目录开始位置相对于archive开始的位移
20	2	.ZIP file comment length(n)	注释长度
22	n	.ZIP Comment	注释内容

0x03:Forensic&Steg

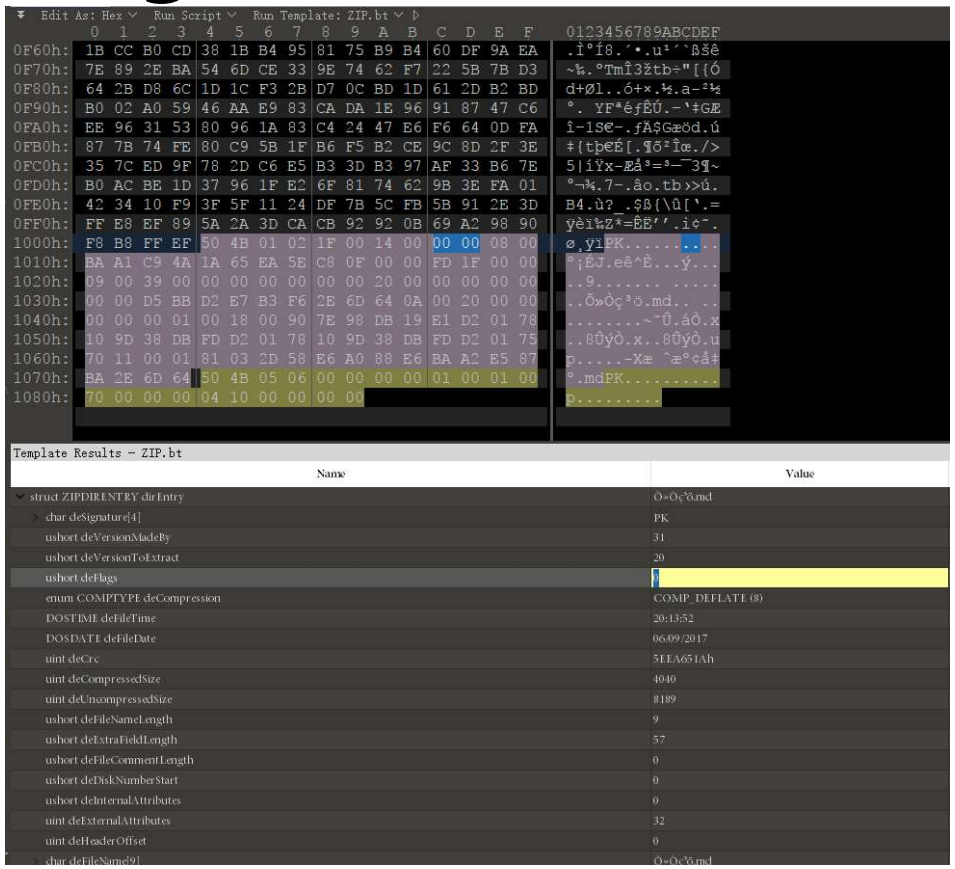
ZIP文件格式：

- 爆破
- 伪加密
- CRC32爆破
- 明文攻击

0x03:Forensic&Steg

ZIP文件格式：

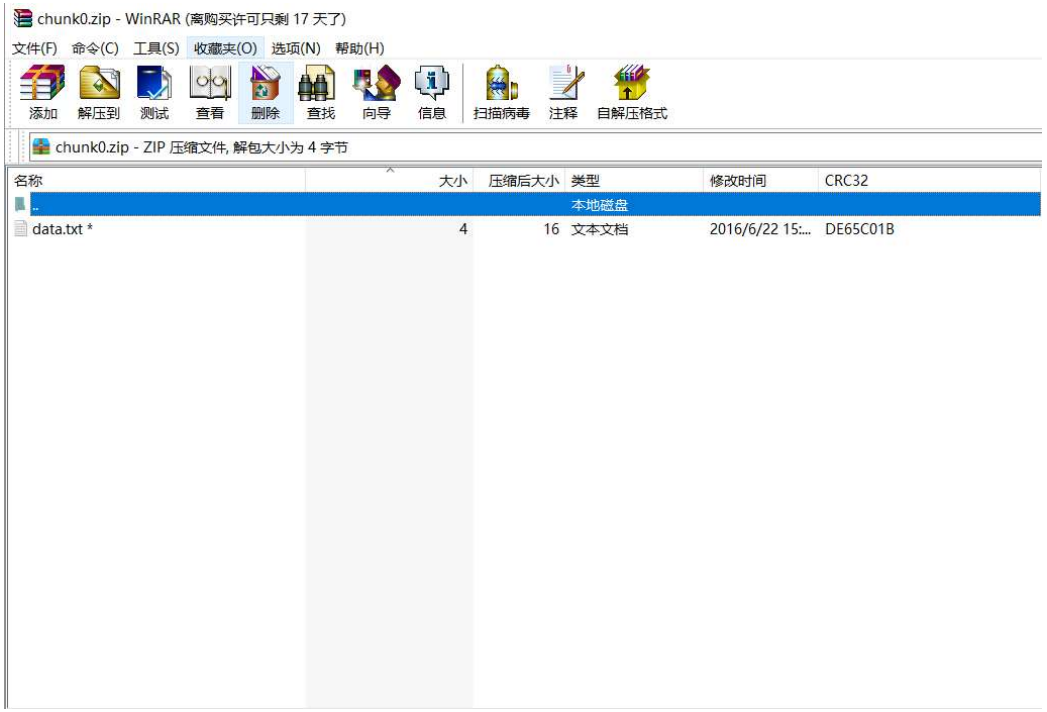
- 爆破
- **伪加密**
- CRC32爆破
- 明文攻击



0x03:Forensic&Steg

ZIP文件格式：

- 爆破
- 伪加密
- **CRC32爆破**
- 明文攻击



0x03:Forensic&Steg

ZIP文件格式：

- 爆破
- 伪加密
- CRC32爆破
- **明文攻击**

0x03:Forensic&Steg

XCTF

流量包：

- 文件修复
- 协议分析
- 数据提取

- Wireshark
- Pcapfix
- Dshell
- tshark

0x03:Forensic&Steg

XCTF

流量包：

文件格式

Global Header	Packet Header	Packet Data	Packet Header	Packet Data	Packet Header	Packet Data	…

0x03:Forensic&Steg

流量包：

文件格式

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;
```

0x03:Forensic&Steg

流量包：

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;
```

0x03:Forensic&Steg

流量包：

协议分析

- 总体把握
  - 协议分级
  - 端点统计
- 过滤赛选
  - 过滤语法
  - Host，Protocol，contains，特征值
- 发现异常
  - 特殊字符串
  - 协议某字段
- 数据提取
  - 字符串取
  - 文件提取

0x03:Forensic&Steg

流量包：

协议分析

TCP等常见

```
------=_Part_2068441_1496292586.1477874934149
Content-Type: application/x-7z-compressed; name=20161008103416509320.7z
Content-Disposition: attachment; filename=20161008103416509320.7z
Content-Transfer-Encoding: base64

N3q8ryccAAQ9AshlwFQFAAAAAAAqAAAAAAAAAOjuWeI83mZQMTuDbrw/ESKoLQCVZl4iDZ8FFWPW
aFcXMf+DsuaUavDOivpta7hQg7PcqY7iN5UfarCBg0a+4A8SbrnwIc/hOZ816qvRJENTXIyyRZUr
fMND3m7XvIbn584uH9EunfOgPjHjUoBxifLmF0hACLdbBAp+RTNFRLFB/q1bo2B/Y2dXsRsfcKgA
p4kJj+A9gfcHgCy99KwqcChWgnvMjbSj9ZsOaHr3ZBN7tY0sJJAJW9K7bnbBSM10bBXd59GWMaiI
UStzW4pc5dulflHwHpDASAMhcO1e5iV3vEIKZix2nfPVed4zQApeG/UccqoEbOB9FElFosOneahz
ICwC/IiYbfye8txOSuhs4lJ1OgirVbS3crSFkorJZcHCqDdXVrtpEJd6yNjzuiXVCJ202BwbWnSz
AtzvtXS+1SKJTnb5/PG8W+XmQz+NMk6ixHSZRCEwLM3Dinu9HJFEv+TFocxWjdrMa0eni8eWrP3b
3ib8v2+8MRLGcTwCv2J27Aurpdo1PG0ST+J6CSh6hdw4ZsQQxmbeWWoLraL6hC4DuKtkLnljuaSr
Jo7FXiSjMz4zRZIRG5MrXYbFV2/KkE5drxXY18WTt6ZtKp6ljoBrT2yGDr2f90I1mfU/o8CrqtYY
m6u8E7mBaYS20gJes2HF8xy3lDGjnRzcMD/KEQvoZTDS/SKa4tmHIahmO3Jj7bZsTl3MJ+CU254D
QvJMa/hJLsej3afAG242nTsV6A0FkYTNepoISWHHiYfdU1mkakAeI97FWiOOANRCnlFnpObKjb8R
0HToJRIr00ZeXmEBOIGAjSxe7bsKoW+DOg1h5fYcrzDxhYSYXTwFFnDzie/s6+ERZOKyZZhhtp8y
AIHaFPF/TEg4BSHbmz2byWOB7sM6gmBwifLEekoAnEbQLAedYnet1EMszgIxuXkj2NHa6uf23Jqj
```

0x03:Forensic&Steg

流量包：

协议分析

TCP等常见

```
------=_Part_2068441_1496292586.1477874934149
Content-Type: application/x-7z-compressed; name=20161008103416509320.7z
Content-Disposition: attachment; filename=20161008103416509320.7z
xxx=%40eval%01%28base64_decode%28%24_POST%5Bz0%5D
%29%29%3B&z0=QGluaV9zZXQoImRpc3BsYXlfZXJyb3JzIiwiMCIpO0BzZXRfdGltZV9saW1pdCgwKTtAc2V0X21hZ2ljX3F1b3Rlc19ydW50aW1lKDApO2VjaG8oIi0%2BfCIpO
zskRj1nZXRfbWFnaWNfcXVvdGVzX2dwYygpP3N0cmlwc2xhc2hlcygkX1BPU1RbInoxIl0pOiRfUE9TVFsiejEiXTskZnA9QGZvcGVuKCRGLCJyIik7aWYoQGZnZXRjKCRmcCkpe
0BmY2xvc2UoJGZwKTtAcmVhZGZpbGUoJEYpO31lbHNle2VjaG8oIkVSUk9SOi8vIENhbiBOb3QgUmVhZCIpO307ZWNobygifDwtIik7ZGllKCk7&z1=%2Fsrv%2FIs-flag-here
%2Fwhoami.jpgHTTP/1.1 200 OK
Server: nginx
Date: Sat, 01 Oct 2016 05:33:36 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: close
Vary: Accept-Encoding
X-Powered-By: PHP/5.4.45

8c65
->|......JFIF.....`.`.....C...........                    .
..
................ $.' ",#..(7),01444.'9=82<.342...C.                    ....

.2!.!22222222222222222222222222222222222222222222222222.....^....".....................................
.....................}........!1A..Qa."q.2....#B...R..$3br.
.....
%&'()*456789:CDEFGHIJSTUVWXYZcdefghijstuvwxyz.....................................................................................
............
AIHaFPF/TEg4BSHbmz2byWOB7sM6gmBwifLEekoAnEbQLAedYnet1EMszgIxuXkj2NHa6uf23Jqj
```

0x03:Forensic&Steg

流量包：

协议分析

TCP等常见

```
    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
  ⌄ [SEQ/ACK analysis]
```

0x03:Forensic&Steg

流量包：

协议分析

其余协议

```
02 0c 20 fc 03 f8 03 47  00 63 ef e6 07 82 1b 5c   .. ..G .c.....\
01 00 15 00 66 00 6c 00  61 00 67 00 2e 00 70 00   ....f.l. a.g...p.
6e 00 67 00 00 c3 00 00  1b 3c 49 1b 3f 89 50 4e   n.g..... .<I.?.PN
47 0d 0a 1a 0a 00 00 00  0d 49 48 44 52 00 00 05   G....... .IHDR...
78 00 00 00 4a 08 03 00  00 00 36 5f cc 04 00 00   x...J... ..6_....
00 5d 50 4c 54 45 4c 69  71 00 00 00 00 00 00 00   .]PLTELi q.......
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00 00 00 fb de 07 17 00  00 00 1e 74 52 4e 53 00   ........ ...tRNS.
df 75 6a 1b ef b3 d0 01  02 15 11 c1 0a 20 04 07   .uj..... ..... ..
2d a6 3c 8c 80 34 60 4d  56 26 99 0d 44 9d bd 4d   -.<..4`M V&..D..M
12 00 00 1a 70 49 44 41  54 78 01 ec d8 e7 8e b3   ....pIDA Tx......
38 14 06 e0 43 77 a1 85  9a 80 e7 dc ff 5d 6e 31   8...Cw.. .....]n1
69 6b 4d 79 3f 47 2c 52  c6 cf 6f eb 34 3b 06 42   ikMy?G,R ..o.4;.B
c1 2b 82 20 08 02 69 d2  ba 36 14 04 41 70 30 55   .+. ..i. .6..Ap0U
d7 ba a5 f7 a7 e3 88 2d  0a 82 20 38 96 60 ab 98   .......- .. 8.`..
35 bd b3 29 e2 4d 91 d0  c1 82 20 08 d6 32 63 2b   5..).M.. .. ..2c+
eb e9 5d e9 ad c5 b2 6f  e8 57 93 8d 78 8f ba df   ..]....o .W..x...
```

0x03:Forensic&Steg

流量包：

协议分析

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

其余协议

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

0x03:Forensic&Steg

XCTF

流量包：

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
```

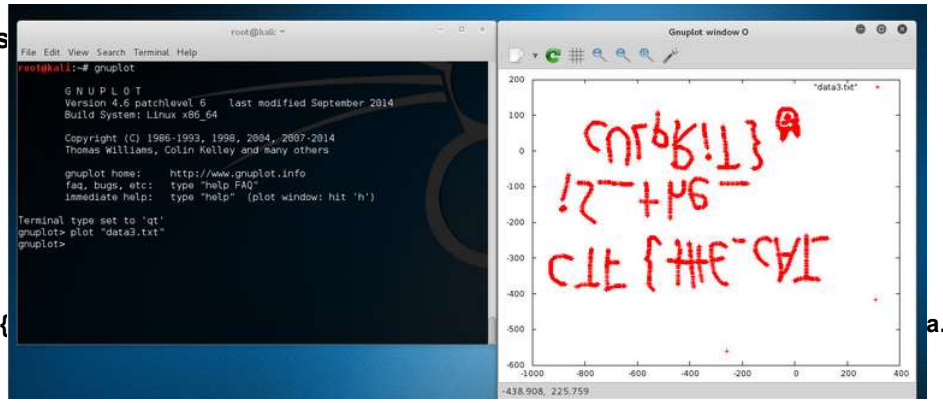
0x03:Forensic&Steg

Google CTF 2016 Forensic-200

文件名：capture.pcapng

what@kali:/tmp$ **tshark -r capture.pcapng -T fields -e us**
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}{** **a.txt > data**
root@kali:~# **tail -n 20 data3.txt**

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

gnuplot> **plot "data3.txt"**

0x03:Forensic&Steg

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}
>>>
```

0x03:Forensic&Steg

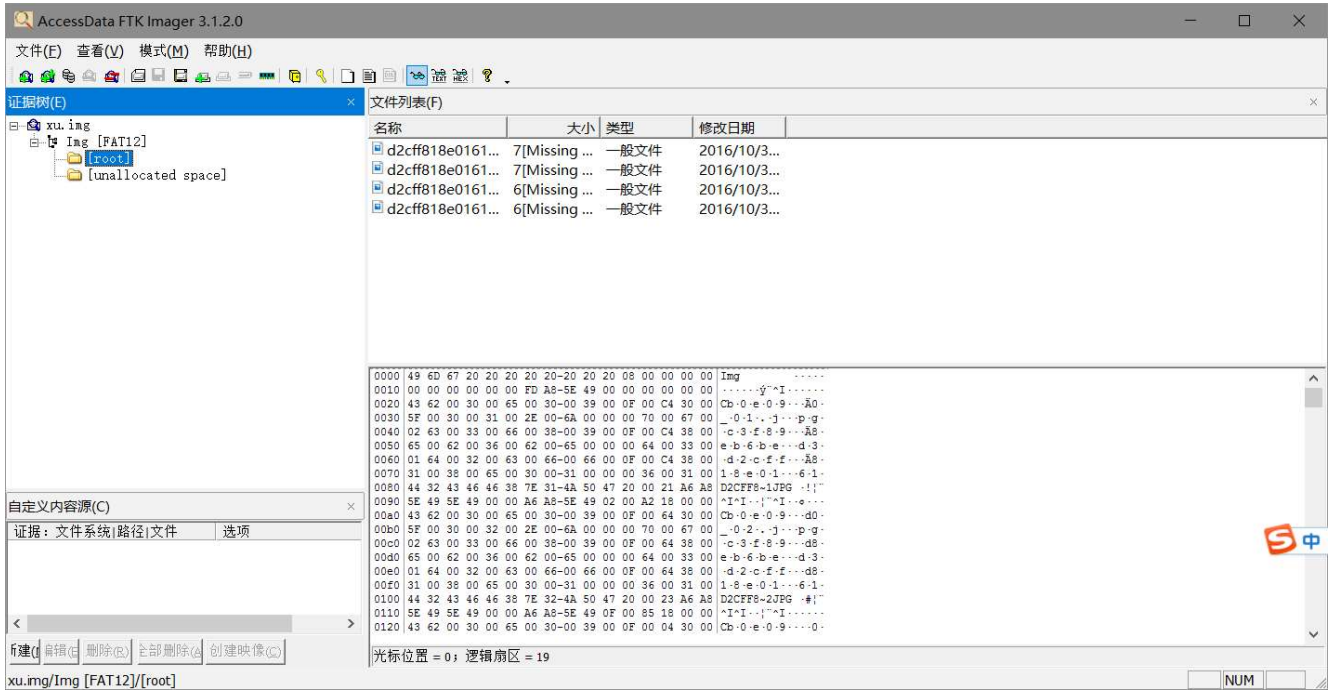
磁盘文件,系统镜像：

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:Forensic&Steg



0x03:Forensic&Steg

内存镜像分析：

1.工具： Volatility
2.确定内存结构
3.查看进程列表
4.根据题目，寻找相关进程，dump目标进程的内存数据进行分析

演示:
Windows内存取证，Volatility的初步使用

0x03:Forensic&Steg

内存镜像分析：

Volatility的初步使用

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）
```

0x03:Forensic&Steg

视频类：

AVI，MP4较为常见

Ffmpeg：

BCTF2016-catvideo

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

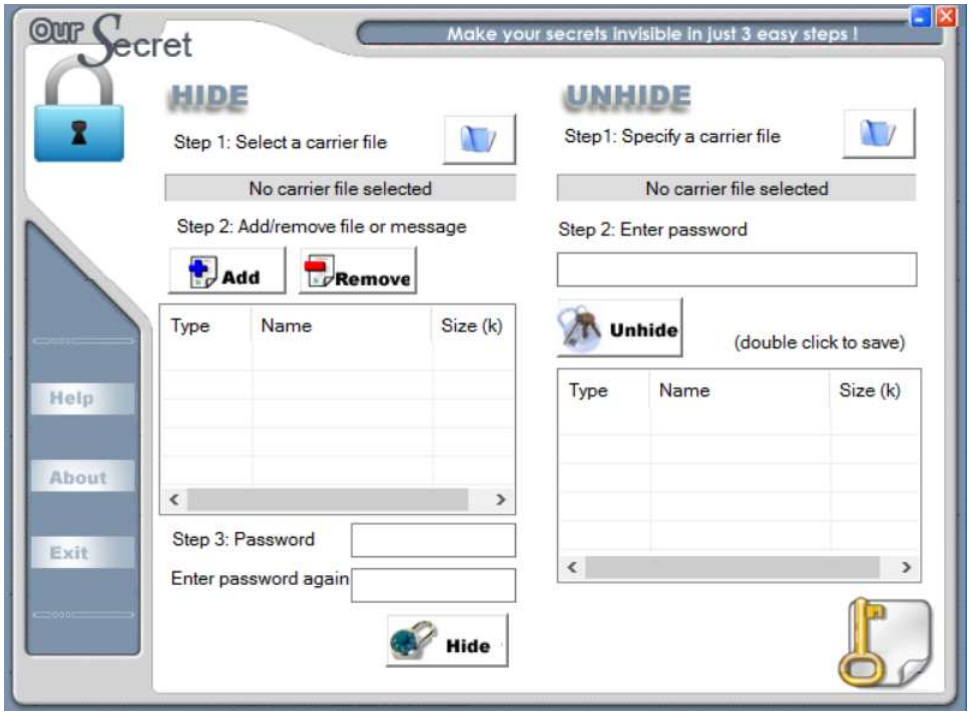
CSAW-CTF-2016-Quals-master\Forensics\Watchword

0x03:Forensic&Steg

视频类：

AVI，MP4较为常见

Ourstrect：



0x03:Forensic&Steg

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:Forensic&Steg

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/snow/snow/snow.html

encode.htm

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	3C	21	44	4F	43	54	59	50	45	20	68	74	6D	6C	3E	09	<!DOCTYPE html>
00000016	20	20	20	20	09	20	09	09	20	09	20	20	09	09	20	20	
00000032	20	20	20	0A	3C	68	74	6D	6C	3E	0A	3C	68	65	61	64	<html> <head
00000048	3E	0A	20	20	20	20	3C	6D	65	74	61	20	63	68	61	72	> <meta char
00000064	73	65	74	3D	22	55	54	46	2D	38	22	3E	0A	20	20	20	set="UTF-8">
00000080	20	3C	6D	65	74	61	20	6E	61	6D	65	3D	22	76	69	65	<meta name="vie
00000096	77	70	6F	72	74	22	20	63	6F	6E	74	65	6E	74	3D	22	wport" content="
00000112	77	69	64	74	68	3D	64	65	76	69	63	65	2D	77	69	64	width=device-wid
00000128	74	68	2C	20	69	6E	69	74	69	61	6C	2D	73	63	61	6C	th, initial-scal
00000144	65	3D	31	2E	30	22	3E	0A	20	20	20	20	3C	6D	65	74	e=1.0"> <met
00000160	61	20	69	64	3D	22	72	65	64	69	72	22	20	68	74	74	a id="redir" htt
00000176	70	2D	65	71	75	69	76	3D	22	72	65	66	72	65	73	68	p-equiv="refresh
00000192	22	20	63	6F	6E	74	65	6E	74	3D	22	31	30	3B	20	75	" content="10; u
00000208	72	6C	3D	68	74	74	70	73	3A	2F	2F	77	77	77	2E	64	rl=https://www.d
00000224	6F	6D	65	6E	65	73	68	6F	70	2E	6E	6F	2F	22	3E	0A	omeneshop.no/">
00000240	20	20	20	20	3C	6C	69	6E	6B	20	72	65	6C	3D	22	64	<link rel="d
00000256	6E	73	2D	70	72	65	66	65	74	63	68	22	20	68	72	65	ns-prefetch" hre

页面 1 / 15 偏移量: 0

origin.htm

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00000000	3C	21	44	4F	43	54	59	50	45	20	68	74	6D	6C	3E	0A	<!DOCTYPE html>
00000016	3C	68	74	6D	6C	3E	0A	3C	68	65	61	64	3E	0A	20	20	<html> <head>
00000032	20	20	3C	6D	65	74	61	20	63	68	61	72	73	65	74	3D	<meta charset=
00000048	22	55	54	46	2D	38	22	3E	0A	20	20	20	20	3C	6D	65	"UTF-8"> <me
00000064	74	61	20	6E	61	6D	65	3D	22	76	69	65	77	70	6F	72	ta name="viewpor
00000080	74	22	20	63	6F	6E	74	65	6E	74	3D	22	77	69	64	74	t" content="widt
00000096	68	3D	64	65	76	69	63	65	2D	77	69	64	74	68	2C	20	h=device-width,
00000112	69	6E	69	74	69	61	6C	2D	73	63	61	6C	65	3D	31	2E	initial-scale=1.
00000128	30	22	3E	0A	20	20	20	20	3C	6D	65	74	61	20	69	64	0"> <meta id
00000144	3D	22	72	65	64	69	72	22	20	68	74	74	70	2D	65	71	="redir" http-eq
00000160	75	69	76	3D	22	72	65	66	72	65	73	68	22	20	63	6F	uiv="refresh" co
00000176	6E	74	65	6E	74	3D	22	31	30	3B	20	75	72	6C	3D	68	ntent="10; url=h
00000192	74	74	70	73	3A	2F	2F	77	77	77	2E	64	6F	6D	65	6E	ttps://www.domen
00000208	65	73	68	6F	70	2E	6E	6F	2F	22	3E	0A	20	20	20	20	eshop.no/">
00000224	3C	6C	69	6E	6B	20	72	65	6C	3D	22	64	6E	73	2D	70	<link rel="dns-p
00000240	72	65	66	65	74	63	68	22	20	68	72	65	66	3D	22	68	refetch" href="h
00000256	74	74	70	73	3A	2F	2F	77	77	77	2E	64	6F	6D	65	6E	ttps://www.domen

0x03:Forensic&Steg

PYC文件类：

字节码修复

```
...
                67      STOP_CODE
                68      STOP_CODE
                69      BINARY_DIVIDE
                70      JUMP_IF_TRUE_OR_POP     5
                73      LOAD_CONST              3: 0
                76      LOAD_CONST              3: 0
                79      BINARY_DIVIDE
Segmentation fault (core dumped)
```

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

0x03:Forensic&Steg

固件分析类：

题目：32C3 CTF 2015 config-bin-150

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 d0  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))
                ]
```

0x03:Forensic&Steg

Pro类：

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）
```

0x03:Forensic&Steg

NES文件类：

NES模拟器直接打开

开金手指

游戏通关获得flag

目录 content

01 Recon

02 Encode

03 Forensic&&Stego

04 Challenge Anlysis

0x04 : Challenge Anlysis

Pretty_Cat (137 solved)	**FlagFlagFlag** (3 solved)	**Green_QRcode** (0 solved)	**SimpleGIF** (11 solved)
128 pt	909 pt	1000 pt	666 pt
1st: yzw	1st: linc		1st: robin123
2nd: HW606	2nd: fydpfg		2nd: acdxvfsvd
3rd: hacker_mao	3rd: zwhubuntu.w>		3rd: h1jack
Pdf_Hack (42 solved)	**ageMagical_Image** (5 solved)	**Tor** (9 solved)	**Hello_XMan** (25 solved)
327 pt	833 pt	714 pt	454 pt
1st: lewis	1st: image	1st: Guf0r	1st: 爱吃姜的萝卜
2nd: linc	2nd: zwhubuntu.w>	2nd: shackle	2nd: Guf0r
3rd: aki	3rd: Binarian	3rd: Levana	3rd: 原来

0x04：Challenge Anlysis

循环爆破

```
#!/usr/bin/env bash

while [ -e *.zip ]; do
   files=*.zip;
   for file in $files;do
      echo -n "Crack ${file}.....";
      output="$(fcrackzip -u -l 1-6 -c '1' *.zip |tr -d '\n')";
      password="${output/PASSWORD FOUND\!\!\!\!: pw == /}";
      if [ -z "${password}" ]; then
         echo "FAIL\!\!\!\!\!";
         break 2;
      fi;
   echo "FOUND PASSWORD : '${password}'";
   unzip -q -P "${password}" "$file";
   rm "${file}";
   done;
done;
```

Thanks for watching 谢谢

版权所有：XCTF