

# ctfshow-Misc入门 图片篇(1-50)

原创

置顶 [z.volcano](#) 于 2021-03-27 14:59:39 发布 13411 收藏 139

分类专栏: [# ctfshow](#) [# 刷题](#) 文章标签: [python](#) [算法](#) [安全](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/weixin\\_45696568/article/details/115261347](https://blog.csdn.net/weixin_45696568/article/details/115261347)

版权



[ctfshow](#) 同时被 2 个专栏收录

9 篇文章 3 订阅

订阅专栏



[刷题](#)

8 篇文章 0 订阅

订阅专栏

八神出的misc入门系列

## 图片篇

图片篇(基础操作)

[misc1](#)

[misc2](#)

[misc3](#)

[misc4](#)

图片篇(信息附加)

[misc5](#)

[misc6](#)

[misc7](#)

[misc8](#)

[misc9](#)

[misc10](#)

[misc11](#)

[misc12](#)

[misc13](#)

[misc14](#)

misc15

misc16

misc17

misc18

misc19

misc20

misc21

misc22

misc23

misc41

#### 图片篇(文件结构)

misc24

misc25

misc26

misc27

misc28

misc29

misc30

misc31

misc32

misc33

misc34

misc35

misc36

misc37

misc38

misc39

misc40

misc42

misc43

misc44

misc45

misc46

misc47

misc48

misc49

#### 图片篇(颜色通道)

misc50

## 图片篇(基础操作)

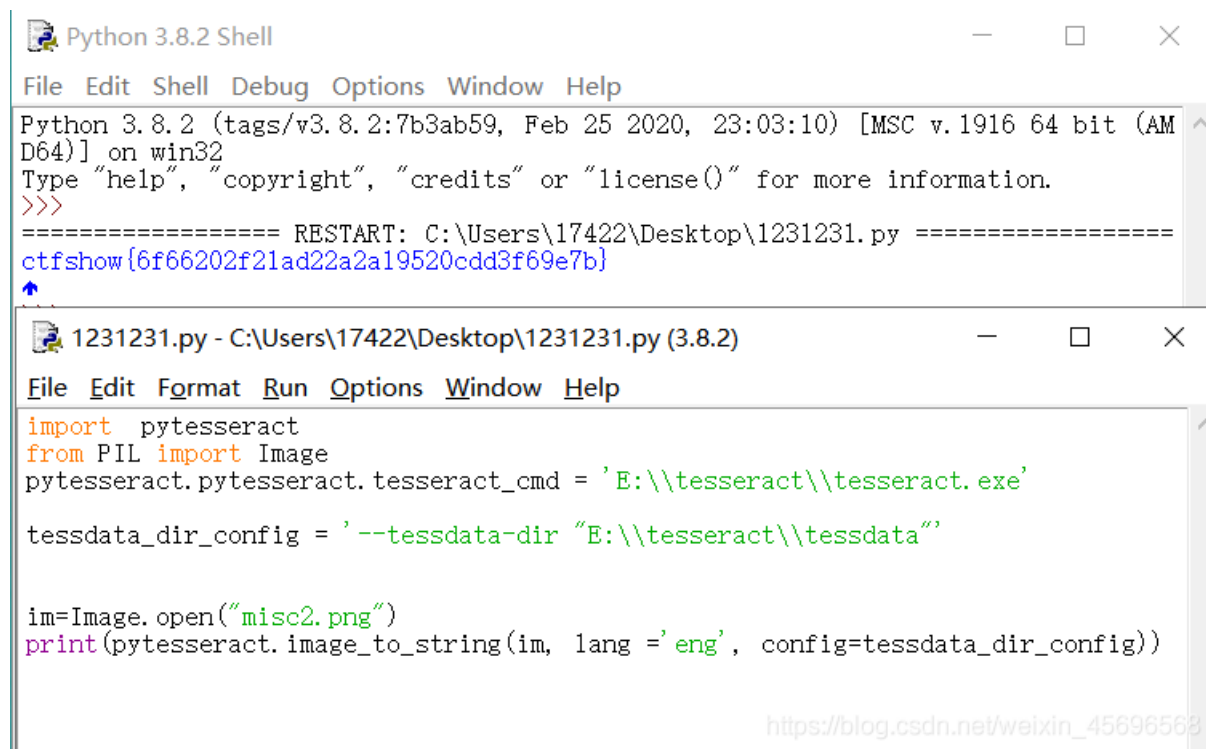
## misc1

打开图片，看到flag

## misc2

下载得到misc2.txt，这是一个点，不要相信题目所给附件的后缀，用winhex打开发现是一个png文件，修改后缀为png，打开看到flag

顺便学了一下Python怎么提取图片中的文字



```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\17422\Desktop\1231231.py =====
ctfshow{6f66202f21ad22a2a19520cdd3f69e7b}
^

1231231.py - C:\Users\17422\Desktop\1231231.py (3.8.2)
File Edit Format Run Options Window Help
import pytesseract
from PIL import Image
pytesseract.pytesseract.tesseract_cmd = 'E:\\tesseract\\tesseract.exe'
tessdata_dir_config = '--tessdata-dir "E:\\tesseract\\tessdata"'

im=Image.open("misc2.png")
print(pytesseract.image_to_string(im, lang='eng', config=tessdata_dir_config))

https://blog.csdn.net/weixin_45696563
```

## misc3

下载得到 [bpg](#) 文件，无法直接查看，到在线网站下载工具

## Download

The following archive contains the source code of the `bpgenc`, `bpgdec` and `bpgview` command line utilities (for Linux) and the source code of the Javascript decoder.

[libbpg-0.9.8.tar.gz](#)

Binary distribution for Windows (64 bit only): [bpg-0.9.8-win64.zip](#)

Unofficial [Github mirror](#).

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

然后查看图片



## misc4

给了6个"txt"文件,分别用winhex打开,查看文件头,发现需要修改后缀。

依次分别改为 `png`、`jpg`、`bmp`、`gif`、`tif`、`webp`。

这题还是要多积累

然后把图片内容拼起来就得到flag。

## 图片篇(信息附加)

## misc5

winhex打开，在尾部看到flag

00000F20	33 3E 20 BA 99 89 97 04 00 00 00 00 49 45 4E 44	3> 00000000 IEND
00000F30	AE 42 60 82 63 74 66 73 68 6F 77 7B 32 61 34 37	@B`,ctfshow{2a47
00000F40	36 62 34 30 31 31 38 30 35 66 31 61 38 65 34 62	6b4011805f1a8e4b
00000F50	39 30 36 63 38 66 38 34 30 38 33 65 7D	906c8f84083e}

## misc6

和上一题差不多，不过把flag藏到了中间，我是用notepad打开，ctrl+f搜索得到的

```
OH NUL STX DC1 ETX ! 1 DC2 EOT Aqag" DC3 ENO2 人 DC4 "B# 罵佯3sb 醜俊CS NAK cs4 xF1% ACK SYN 2. x6  
: 誦c 欄2 遑羨 xB5 DC2 弘 誦c +`
```

```
JE NUL NUL NUL NUL Img · NUL NUL NUL SI ctfshow{d5e937aefb091d38e70d927b80e1e2ea} NUL SOH NUL  
sMbool NUL NUL NUL NUL NUL CrnCbool NUL NUL NUL NUL NUL CntCbool NUL NUL NUL NUL NUL Lblsbool  
NUL NUL NUL SO cropRectBottomlong NUL NUL NUL NUL NUL NUL NUL NUL cropRectLeftlong NUL NUL NUL
```

```
L SOH 8BIME TX xF8 NUL NUL NUL NUL NUL NUL NUL NUL xFF xFF xFF xFF xFF xFF xFF xFF xFF xFF xFF xFF
```

```
JE NUL NUL SOH NUL NUL NUL NUL NUL NUL null TEXT NUL NUL NUL SOH NUL NUL NUL NUL NUL NUL NUL Msge TEX  
SOH NUL NUL NUL xA0 NUL NUL NUL ESC NUL NUL SOH xE0 NUL NUL 2 xA0 NUL NUL EOT xD3 NUL CAN NUL SO
```

查找 替换 文件查找 标记

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

## misc7

这个题给提示了：flag在图片文件信息中。

直接右键查看属性是常用的方法，不过获取不到图片的全部文件信息，也得不到这题的flag

使用[在线网站](#)查看详细的exif信息

用notepad也能做

```
v* 蕪舜鹽b 辮] 城 ctfshow{c5e77c9c289275e3f307362e1ed86bb7} | v* 蕪舜鹽b 辮] 如
```

## misc8

用winhex打开，发现图片中隐写了其他图片，手动或者binwalk或者foremost分离出图片。

```

00000EE0 10 02 00 00 77 7A 20 07 00 00 00 07 10 00 00 00
00000EF0 10 29 81 10 00 00 20 52 02 21 00 00 40 A4 04 42
00000F00 00 00 80 48 09 84 00 00 00 91 12 08 01 00 00 22
00000F10 25 10 02 00 00 44 4A 20 04 00 00 88 52 08 FF 07
00000F20 33 3E 20 BA 99 89 97 04 00 00 00 00 49 45 4E 44
00000F30 AE 42 60 82 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D
00000F40 49 48 44 52 00 00 03 84 00 00 00 96 08 02 00 00
00000F50 00 09 DA D1 61 00 00 00 09 70 48 59 73 00 00 12
00000F60 74 00 00 12 74 01 DE 66 1F 78 00 00 07 98 69 54
00000F70 58 74 58 4D 4C 3A 63 6F 6D 2E 61 64 6F 62 65 2E
00000F80 78 6D 70 00 00 00 00 00 3C 3F 78 70 61 63 6B 65
00000F90 74 20 62 65 67 69 6E 3D 22 EF BB BF 22 20 69 64
00000FA0 3D 22 57 35 4D 30 4D 70 43 65 68 69 48 7A 72 65
) R ! @ B
€H " \ "
% DJ ^R ŷ
3> 0™%- IEND
@B`,%PNG
IHDR " -
ÚNa pHYs
t t Pf x ~iT
XtXML:com.adobe.
xmp <?xpacke
t begin="i>¿" id
="W5M0MpCehiHzre

```

查看图片得到flag

## misc9

提示：flag在图片块里。

```

> struct PNG_SIGNATURE sig 0h 8h Fg: Bg:
> struct PNG_CHUNK chunk[0] IHDR (Critical, Public, Unsafe to Copy) 8h 19h Fg: Bg:
> struct PNG_CHUNK chunk[1] pHYs (Ancillary, Public, Safe to Copy) 21h 15h Fg: Bg:
> struct PNG_CHUNK chunk[2] iTXt (Ancillary, Public, Safe to Copy) 36h 528h Fg: Bg:
> struct PNG_CHUNK chunk[3] tEXt (Ancillary, Public, Safe to Copy) 55Eh 3Dh Fg: Bg:
  uint32 length 49 55Eh 4h Fg: Bg:
  union CTYPE type tEXt 562h 4h Fg: Bg:
> struct PNG_CHUNK_TEXT text Warning = ctfshow{5c5e819508a3ab1fd823f11e83e93c75} 566h 31h Fg: Bg:
  uint32 crc 6A940E9h 597h 4h Fg: Bg:
> struct PNG_CHUNK chunk[4] IDAT (Critical, Public, Unsafe to Copy) 59Eh B7Fh Fg: Bg:
  uint32 length 2931 59Eh 4h Fg: Bg:
  union CTYPE type IDAT 59Fh 4h Fg: Bg:
  ubyte data[2931] 5A3h B73h Fg: Bg:
  uint32 crc C464AE32h 1116h 4h Fg: Bg:
> struct PNG_CHUNK chunk[5] IEND (Critical, Public, Unsafe to Copy) 111Ah 4h Fg: Bg:

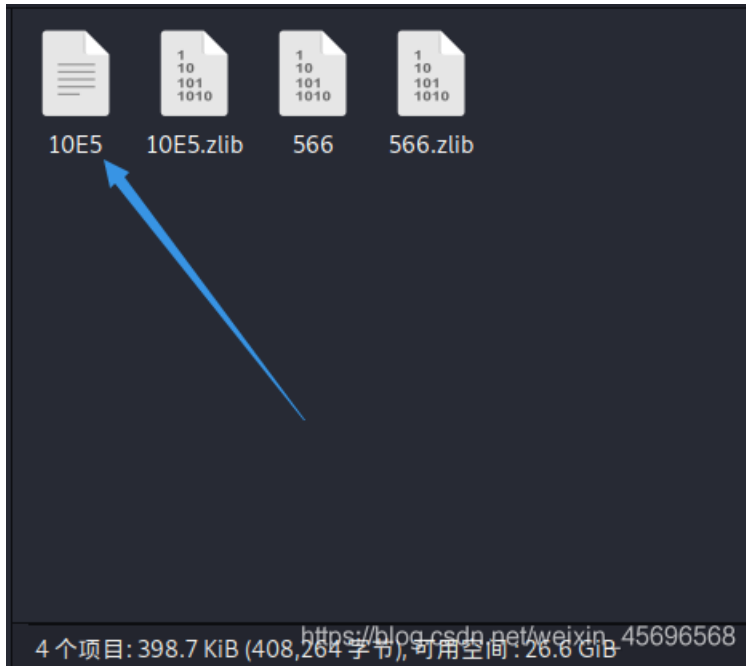
```

用misc6的方法也能做出来，我这里是用的010editor做的。

## misc10

提示：flag在图片数据里。

binwalk -e 分离出数据后



打开第一个文件得到flag

关于原理，八神在群里说过，我忘记截图了，这里用套神师傅博客里的图



LV53 8神 cheyenne 🌟 🍷

沐秋的清晨 下午6:24

阿这

来自其他聊天

@沐秋的清晨 binwalk可以一把梭，是因为binwalk会找到zlib块的标记然后提取出来，同时因为这是个压缩数据，binwalk的-e参数会自动把提取到的压缩包尝试进行解压，所以最后的提取结果里就有原始的那段文本，就是flag了

[https://pic9.bcebos.com/welkin\\_48896568](https://pic9.bcebos.com/welkin_48896568)

## misc11

提示：flag在另一张图里。

这个图有两个IDAT块，而且没有隐写其他的数据

> struct PNG_SIGNATURE sig		0h	8h	Fg:	Bg:
> struct PNG_CHUNK chunk[0]	IHDR (Critical, Public, Unsafe to Copy)	8h	19h	Fg:	Bg:
▼ struct PNG_CHUNK chunk[1]	IDAT (Critical, Public, Unsafe to Copy)	21h	B7Fh	Fg:	Bg:
uint32 length	2931	21h	4h	Fg:	Bg:
> union CTYPE type	IDAT	25h	4h	Fg:	Bg:
> ubyte data[2931]		29h	B73h	Fg:	Bg:
uint32 crc	C464AE32h	B9Ch	4h	Fg:	Bg:
▼ struct PNG_CHUNK chunk[2]	IDAT (Critical, Public, Unsafe to Copy)	2A0h	1D81h	Fg:	Bg:
uint32 length	7541	BA0h	4h	Fg:	Bg:
> union CTYPE type	IDAT	BA4h	4h	Fg:	Bg:
> ubyte data[7541]		BA8h	1D75h	Fg:	Bg:
uint32 crc	228B674Bh	291Dh	4h	Fg:	Bg:
> struct PNG_CHUNK chunk[3]	IEND (Critical, Public, Unsafe to Copy)	2B0h	4h	Fg:	Bg:

应该是上次八神的PNG隐写入门赛的png14一个思路，试着把第一个IDAT块的数据删除，然后另存为一张新图片，这个过程可以手动操作，也可以使用tweakpng工具。



flag: `ctfshow{44620176948fa759d3eeafeac99f1ce9}`

## misc12

提示: flag在另一张图里。

和上题一样的提示, 所以思路是一样的。

不过这题有30个IDAT块, 用PNGDebugger跑了一下, 发现没有出错的IDAT块...

慢慢试吧

测试后发现需要删掉前8个IDAT块

flag: `ctfshow{10ea26425dd4708f7da7a13c8e256a73}`

## misc13

提示: flag位置在图片末尾。

看到提示, 第一反应是notepad++打开, ctrl+f搜索ctfshow, 无果...

老老实实用winhex打开, 看尾部数据, 发现两坨可疑字符串, 注意前面那一坨得到的才是正确flag, 我复现的时候错用后一段了, 导致得到错误的flag

00000DC0	D5	45	CE	F3	E0	DC	00	5D	04	5C	17	25	5B	7C	90	09	0E10a0	-u\	8,	1-
00000DD0	82	F9	B1	57	49	EF	33	40	09	C8	0B	C6	2B	E4	02	3A	,	ù±WII3@	È	Æ+ä :
00000DE0	D4	63	1A	74	B9	66	85	73	86	68	AA	6F	4B	77	B0	7B	Öc	t¹f...sth	ª	oKw°{
00000DF0	21	61	14	65	53	36	A5	65	54	33	34	65	78	61	25	34	!a	eS6¥eT34exa%4		
00000E00	DD	38	EF	66	AB	35	10	31	95	38	1F	62	82	37	BA	65	Ý8if«5	1•8	b,7°e	
00000E10	45	34	7C	32	54	64	7E	37	3A	64	E4	65	F1	36	FA	66	E4 2Td~7:däeñ6úf			
00000E20	F5	34	1E	31	07	32	1D	66	54	38	F1	33	32	39	E9	61	ø4	1	2	fT8ñ329éa
00000E30	6C	7D	94	28	62	E7	A1	CA	A7	24	8E	7E	B8	2A	AC	1F	l}"	(bç;	ÊS\$Ž~,	*¬
00000E40	A1	93	E3	FF	9F	13	00	AF	30	88	2A	73	79	F6	9F	49	i"	äyÿ	0^*	syöÿI
00000E50	20	D1	85	84	93	13	F7	35	D1	85	25	55	17	06	9E	EA	Ñ...,"	÷5Ñ...*	U	žè
00000E60	B9	59	9C	C7	15	3F	79	B2	A6	4D	C3	17	AA	7C	12	31	¹YæÇ	?y²!	MÃ	ª
00000E70	25	03	FE	FE	AB	C8	63	7C	BE	CE	1C	DB	4E	D4	7D	35	%	þþ«Èc	¾Î	ÛNÔ}5
00000E80	D6	43	BD	B3	FF	7C	5C	1A	78	1B	7F	02	6C	79	53	32	ÖC½³ÿ \	x	lyS2	
00000E90	7A	7C	C4	3E	97	2E	74	B2	47	17	54	C1	A6	E5	6F	ED	z Ä>-	.t²G	TÄ!	áoí
00000EA0	38	C5	C8	0F	49	89	93	39	04	D5	A7	DF	27	14	58	9C	8ÅÈ	I%~9	ÕSß'	Xæ
00000EB0	96	4C	1F	5B	DF	9C	92	92	39	AB	A4	3B	D3	CA	31	09	-L	[Bæ'	'9«ª;	ÓÊ1
00000EC0	C0	59	EA	F3	0F	5A	23	DC	DC	34	C8	DE	3A	9C	35	A0	ÀYèó	Z#ÜÜ4ÈE:	æ5	
00000ED0	A7	AB	D5	56	45	BC	5D	3F	54	50	D2	40	DD	B6	14	7D	S«ÖVE¾	]?	TPÒ@	ÿŸ }
00000EE0	FC	DC	FE	33	D2	72	35	C0	72	BB	97	92	BE	5C	89	23	üÛþ3Ör5Är»-	'¾\	%#	
00000EF0	88	B8	53	8D	17	F3	F9	63	1A	74	B9	66	85	73	86	68	^,	S	óuc	t¹f...sth
00000F00	AA	6F	4B	77	B0	7B	21	61	14	65	53	36	A5	65	54	34	ª	oKw°{	!a	eS6¥eT4
00000F10	34	36	78	63	25	34	DD	38	EF	66	AB	37	10	33	95	39	46xc%4Ý8if«7	3•9		
00000F20	1F	62	82	37	BA	65	45	62	7C	32	54	64	7E	31	3A	64	b,7°eEb 2Td~1:d			
00000F30	E4	65	F1	36	FA	65	F5	34	1E	31	07	32	1D	66	54	38	äeñ6úeø4	1	2	fT8
00000F40	F1	33	32	39	E9	61	6C	7D	2B	F5	E0	D5	3E	44	E6	CD	ñ329éal}	+øàÖ>	Dæí	
00000F50	C8	C8	F3	A5	2F	79	33	96	FF	41	76	F9	6F	49	F4	BA	È.È	¾/v3-bAvünIª		

注意到 { 前面那一串字符, 从第一位开始, 每隔一位选取一个字符, 连起来就是ctfshow

这里把这串十六进制数值复制下来, 按照规律选取正确的数值

```
s="631A74B96685738668AA6F4B77B07B216114655336A5655433346578612534DD38EF66AB35103195381F628237BA6545347C3254647E373A64E465F136FA66F5341E3107321D665438F1333239E9616C7D"
flag=""
for i in range(0,len(s),4):
    flag += s[i]
    flag += s[i+1]
print(flag)
```

然后十六进制转字符，得到flag: `ctfshow{ae6e3ea48f518b7e42d7de6f412f839a}`

## misc14

提示: flag在那张图里。

binwalk分析发现有额外数据，直接binwalk -e或者foremost分离不出来，无奈手撸

```
(volcano@kali)-[~]
└─$ binwalk /home/volcano/桌面/misc14.jpg
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	JPEG image data, EXIF standard
12	0xC	TIFF image data, big-endian, offset of first image directory: 8
1681	0x691	TIFF image data, big-endian, offset of first image directory: 8
2103	0x837	JPEG image data, JFIF standard 1.01

从选中的数据开始，复制到结尾，新建为一个jpg文件

000007E0	00 01 00 01 1B 00 00 00 00 00 00 01 00 02 00 00 02 01 00 04 00	(
00000800	00 00 01 00 00 01 A6 02 02 00 04 00 00 00 01 00	;
00000810	00 04 D5 00 00 00 00 00 00 00 00 48 00 00 00 01 00	õ
00000820	00 00 48 00 00 00 01 FF D8 FF E0 00 10 4A 46 49	H
00000830	46 00 01 01 01 00 78 00 78 00 00 FF DB 00 43 00	H yøÿà JFI
00000840	02 01 01 02 01 01 02 02 02 02 02 02 02 02 03 05	F x x yÛ C
00000850	03 03 03 03 03 06 04 04 03 05 07 06 07 07 07 06	
00000860	07 07 08 09 0B 09 08 08 0A 08 07 07 0A 0D 0A 0A	
00000870	0B 0C 0C 0C 0C 07 09 0E 0E 0D 0C 0E 0B 0C 0C 0C	

拿到flag: `ctfshow{ce520f767fc465b0787cdb936363e694}`

## misc15

提示: flag被跳过去了。

winhex打开直接看到flag

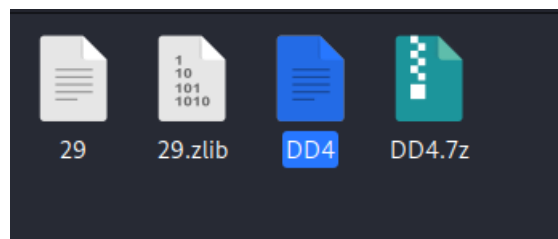
```
ANSI ASCII
00 BMN g (
00 " -
00 ø t t
80 € €
80 €€ € € € €€
FF €€€ ÀÀÀ ÿ ÿ
FF ÿÿ ÿ ÿ ÿ ÿÿ
34 ÿÿÿ txo=+) bM4
75 DSyi$;U7(FT-Eufu
4E VmRt8c/q5LRQsdCN
3E Vhi!O?Ij) ,IH8u>
71 %lMh}C vs1vt,p(q
3E JKN I/^%h:v-b}>
6F IYtj!qa3 ectfsho
36 w{fbe7bb657397e6
34 e0a6adea3e402654
67 25}P[ PBxM1 KDFg
31 b<bWPF919k{\10<1
5F ba{c cwqIZ_Yk.g_
38 E<IhZIWznCZm>)Y8
63 M|c Y.A%hj&j>,Yc
48 *yxKvRg|#%"LT/H
```

八神的题肯定没有这么简单，不过我没想到预期解是怎样的

## misc16

提示: flag在图片数据里。

binwalk -e, 然后打开这个文件得到flag, 原理和misc10一样(我猜的)



## misc17

提示: flag在图片数据里。

先binwalk分析，没啥问题，然后试试zsteg

```
(volcano@kali)-[~]
└─$ zsteg /home/volcano/桌面/misc17.png
[?] 3544 bytes of extra data after zlib stream
extradata:0
00000000: e1 1f 30 53 86 4f c5 a4 1b f5 e6 e5 c7 46 0a 92 |..0S.0.....F..
00000010: 9b ee 72 e7 c9 9e b9 a7 74 de 92 4d ad 61 5b 58 |..r.....t..M.a[X
00000020: f2 98 65 77 2b d2 d3 85 32 fc 08 83 86 1f 0f 1e |..ew+...2.....
00000030: cb ab ac 9c 4b ca 02 20 e2 ce e4 ae 60 1a 2c c6 |...K.. .....,
00000040: 7b c8 9a 77 31 2f 9e 67 db d9 3e 53 fe 17 a5 50 |{..w1/.g..>S...P
00000050: 20 e5 1d 8c d5 49 4e 52 a5 54 31 cb 8b c5 3b 09 |...INR.T1...;.
00000060: a2 a6 fe 5b da 4f 9e 78 9c 5d 46 d6 e2 6b 6b 2a |...[.0.x.]F..kk*
00000070: f2 62 0c ba 70 19 a0 27 f3 84 77 99 02 77 05 79 |.b..p.. '..w..w.y
00000080: 5b 44 b7 79 b3 54 11 a1 f3 54 34 56 7e ff 55 d1 |[D.y.T...T4V~.U.
00000090: c6 39 90 c8 21 7f 26 39 44 58 78 c3 ed 37 4a 7c |.9..!.69DXx..7J|
000000a0: 50 24 e8 79 7b 4b 9c fa 2a 2c bb e8 b9 fb 40 2c |P$.y{K..*,...@,
000000b0: 50 05 21 4c 3b 29 65 b4 60 1c 27 bb 4c 16 bf f1 |P.!L;)e.`'.L...
000000c0: 77 c0 55 04 5e 25 0e 18 1e 58 ab 0f 13 11 f2 3f |w.U.^%...X.....?
000000d0: cf a0 32 b1 f5 a8 1b 99 a7 4b 46 89 cf 85 89 50 |..2.....KF....P
000000e0: 88 20 8f 4f fd e2 97 55 68 73 b4 96 ba dd 25 a3 |..0...Uhs...%
000000f0: 83 72 3f 99 77 9e 0a 08 50 4f 11 8f 87 65 e0 29 |.t%w...p0...e..)
https://blog.csdn.net/weixin_45696568
```

发现隐藏的数据，位置处于 `extradata:0`

将数据提取出来: `zsteg -E "extradata:0" /home/volcano/桌面/misc17.png > 1.txt`

然后再binwalk -e把1.txt中的数据分离出来，拿到flag

---

ctfshow{0fe61fc42e8bbe55b9257d251749ae45}

misc18

提示: flag在标题、作者、照相机和镜头型号里。

## IFD0

型号	28ac17e5f0
创作者	5d60c208f7
XP标题	ctfshow{32
XP作者	5d60c208f7
Padding	(Binary data 2072 bytes, use -b option to extract)

## ExifIFD

Padding	(Binary data 2060 bytes, use -b option to extract)
---------	--

## XMP-rdf

About	uuid:faf5bdd5-ba3d-11da-ad31-d33d75182f1b
-------	---

## XMP-dc

标题	ctfshow{32
描述	ctfshow{32
Creator	5d60c208f7



## XMP-microsoft

镜头型号	2d4cf5a839}	<a href="https://blog.csdn.net/weixin_45696568">https://blog.csdn.net/weixin_45696568</a>
------	-------------	---

## misc19

提示: flag在主机上的文档名里。

## IFD0

子文件类型	Full-resolution image
图像宽度	900
图像高度	150
采样位数	8 8 8
压缩	LZW
PhotometricInterpretation	RGB
文档名称	ctfshow{dfdcf08038cd446a5} 
Strip偏移	21688 25422
方向	Horizontal (normal)
SamplesPerPixel	3
RowsPerStrip	97
Strip字节数s	3733 749
X分辨率	72
Y分辨率	72
PlanarConfiguration	Chunky
分辨率单位	inches
软件	Adobe Photoshop CC 2019 (Windows)
修改日期	2021:03:25 10:35:18
主机	eb50782f8d3605d} 
预测	Horizontal differencing

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

## misc20

提示: flag在评论里。

到前面用过的exif信息查看在线网站，上传图片，看到信息(谐音可还行)

File	
FileType	JPEG
FileTypeExtension	jpg
MIMEType	image/jpeg
ExifByteOrder	Big-endian (Motorola, MM)
Comment	这图片也太难看了。来自：西替爱抚秀大括号西九七九六四必一诶易西爱抚零六易一弟七九西二一弟弟诶弟五九三易四二大括号
ImageWidth	900
ImageHeight	150
EncodingProcess	Baseline DCT, Huffman coding
BitsPerSample	8
ColorComponents	3
YCbCrSubSampling	YCbCr4:2:0 (2 2)

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

ctfshow{c97964b1aecf06e1d79c21ddad593e42}

## misc21

提示：flag在序号里。

ExifIFD	
Exif版本	0232
ComponentsConfiguration	Y, Cb, Cr, -
SecurityClassification	Top Secret
Flashpix版本	0100
色彩空间	Uncalibrated
序列号	686578285826597329

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

转字符得到 `hex(X&Ys)`

Input
686578285826597329

## Output

hex(X&Ys)

发现上面有两组与XY有关的数据，中间还有https://ctf.show/和ctfshow{}，应该是八神的贴心小提示，怕我们萌新找不对地方(上次做这题的时候就没注意到...)

### IFD0

X分辨率	3902939465
Y分辨率	2371618619
PageName	https://ctf.show/
X定位	1082452817
Y定位	2980145261
目标Printer	ctfshow{}

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

根据提示 `hex(X&Ys)`，应该是要把这里的十进制数值转为十六进制，我最开始是把四段拼起来得到 `3902939465237161861910824528172980145261`，然后转十六进制，再套上 `ctfshow{}`，然后错了...

如果不是整体直接转换的话，那么应该就是每段分别转hex，然后拼起来

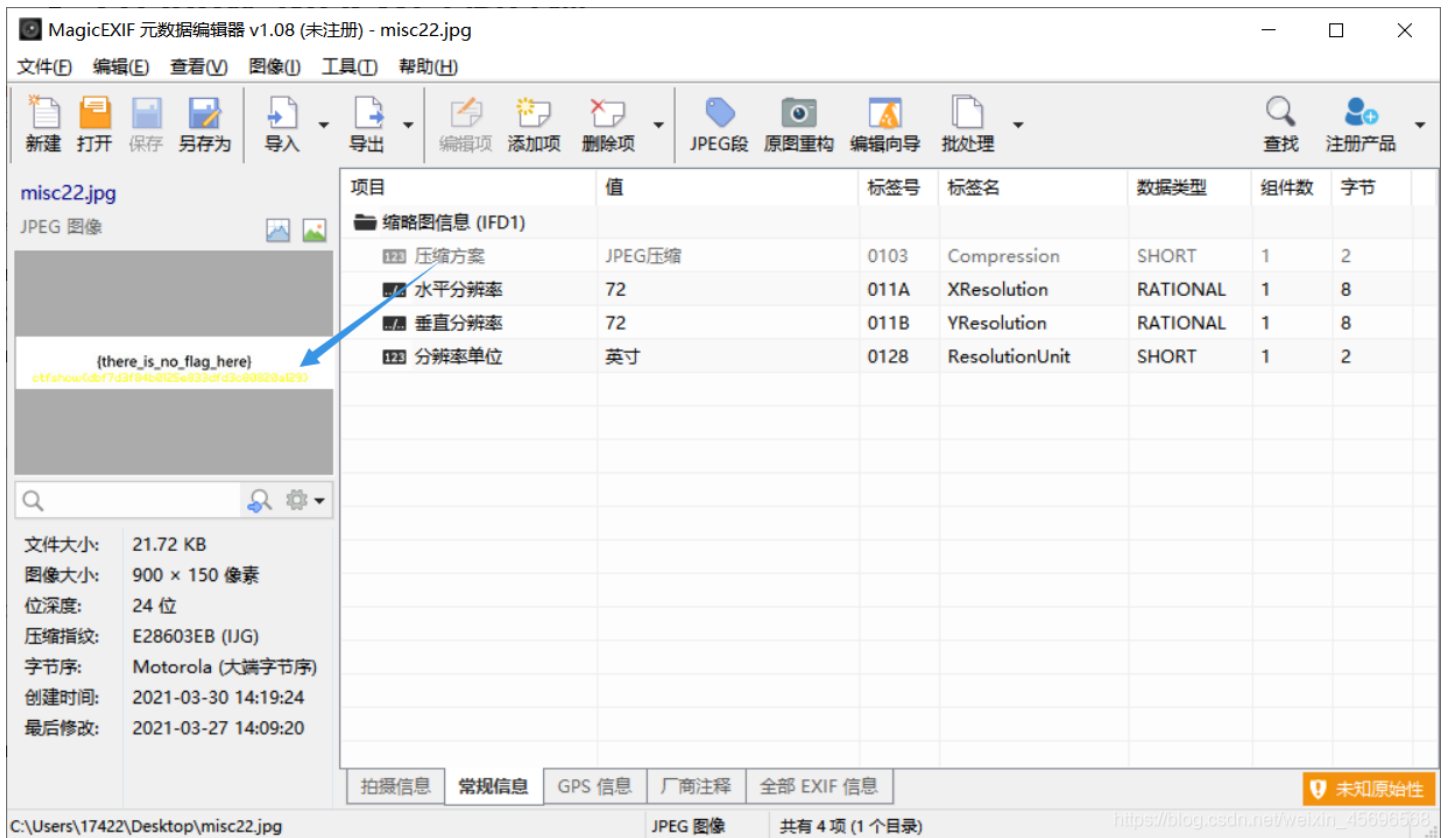
最终得到: `ctfshow{e8a221498d5c073b4084eb51b1a1686d}`

## misc22



提示: flag在图片里。

用magicexif打开, 直接发现flag



原来八神是把flag藏到了缩略图中



我换了个颜色, 方便确认

我第一遍看出来的是ctfshow{dbf7d3f84b0125e833dfd3c80820a129}

然后提交一直失败，无奈去问八神，原来这个不是o，是c



所以 `ctfshow{dbf7d3f84b0125e833dfd3c80820a129}`

## misc23

提示：flag在时间里。

21题搞出来了，这题就有思路了

使用exiftool看一下发现有好几个历史时间，上面的 `History Action` 中有提示

```
Modify Date      : 2021:03:25 16:02:50+08:00
Document ID     : xmp.did:49520599-6932-e144-8f4b-dfd5873be5bc
History Action  : ctfshow{}, UnixTimestamp, DECtoHEX, getflag
History Instance ID : xmp.iid:1, xmp.iid:2, xmp.iid:3, xmp.iid:4
History Software Agent : Adobe Photoshop CC 2019 (Windows), Adobe Photoshop CC 2019 (Windows), Adobe Photoshop CC 2019 (Windows)
History When    : 1997:09:22 02:17:02+08:00, 2055:07:15 12:14:48+08:00, 2038:05:05 16:50:45+08:00, 1984:08:03 18:41:46+08:00
History Changed  : /
```

Timestamp指的是时间戳，那个前缀没搞懂是什么意思，DECtoHEX是十进制转十六进制

这里利用在线网站获取时间戳

现在: **1617089860**      控制: ■ 停止

时间戳  秒(s)   北京时间

时间  北京时间   秒(s)

最后得到4段

```
874865822 2699237688 2156662245 460377706
```

按照21题的经验，分别hex后拼在一起

得到: `ctfshow{3425649ea0e31938808c0de51b70ce6a}`

## misc41

提示:

```
H4ppy Apr1l F001's D4y!
愚人节到了，一群笨蛋往南飞，一会儿排成S字，一会儿排成B字。
```

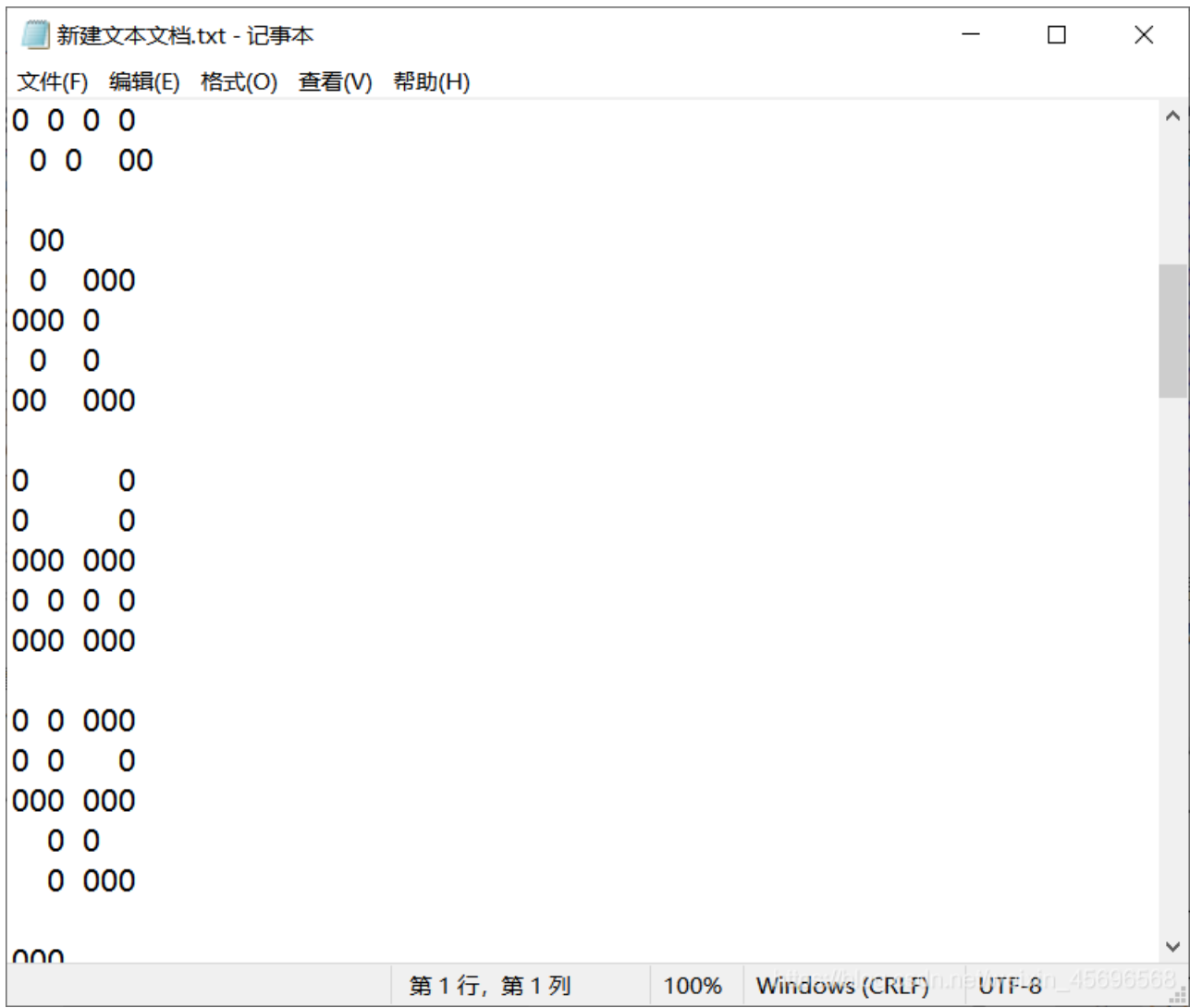
愚人节限定题，下载得到misc41.jpg，用winhex打开，发现是jpg的文件尾，但是文件头对不上，本来想以这个方向为突破点，没得到结果...

后面看了套神的wp才知道，提示中的第二句说的就是我...

第一句提示的 **F001** 才是真突破点，这个位置有大量F001，看起来组成了某种形状

00002DE0	66 51 74 AD C0 42 62 D4 F5 CA 15 D5 6A F9 F9 29	fQt-ÀBbÔðÊ Òjùù)
00002DF0	F0 01 F0 01 F0 01 46 DF F0 01 F0 01 F0 01 92 46	ð ð ð Fßð ð ð ' F
00002E00	F0 01 4D BB 67 07 AB 47 2A 10 35 10 F0 01 89 AA	ð M»g «G* 5 ð %ª
00002E10	F0 01 F0 01 F0 01 BA 7F F0 01 F0 01 F0 01 F6 38	ð ð ð ° ð ð ð ö8
00002E20	CF FE 3B 1E F0 01 F4 40 F0 01 B1 EF 6C 90 DA F3	Ïp; ð ðÊð ±il Úó
00002E30	F0 01 F0 01 F0 01 73 AB F0 01 F0 01 F0 01 AD C8	ð ð ð s«ð ð ð -È
00002E40	75 3D F7 9F D6 36 3E FE 36 42 38 D7 9A A4 39 61	u=÷ÿÖ6>þ6B8×šª9a
00002E50	C5 E5 F0 01 F0 01 46 D3 71 88 F0 01 7F E6 3B BB	Ååð ð FÓq^ð æ;»
00002E60	EF 6B F0 01 EF C4 C0 66 F0 01 F0 01 FF 50 8B 14	ikð iÄÀfð ð ýP<
00002E70	F0 01 F0 01 F0 01 67 86 DC 31 F0 01 EA 4B 99 4A	ð ð ð g†Ülð êK™J
00002E80	24 CA F0 01 C3 93 96 97 82 B8 F0 01 56 74 04 FD	ŞÊð Å"--, ð Vt ý
00002E90	F0 01 F0 01 D0 EF 23 5B F0 01 F0 01 F0 01 79 19	ð ð Ði#[ð ð ð y
00002EA0	E2 94 95 0A 71 3D 17 87 4B F6 F2 A6 EF 88 32 09	â"• q= †Kðð!i^2
00002EB0	2B 4B F0 01 5C 9F D6 4F F0 01 99 FF F0 01 DB 58	+Kð \ÿÖð ð ðÿð ÛX
00002EC0	F0 01 F0 01 26 8D 92 39 F0 01 76 92 F0 01 FD 25	ð ð & '9ð v'ð ý%
00002ED0	77 07 F0 01 1F D6 B1 82 F0 01 F0 01 F0 01 5E D3	w ð Ö±,ð ð ð ^Ó
00002EE0	B0 03 F0 01 3B F4 A6 35 B4 E5 F7 5D F0 01 4A D1	° ð ;ð!5'á÷]ð JÑ
00002EF0	F0 01 F0 01 F0 01 09 EB 41 13 3C 23 F0 01 12 7E	ð ð ð ëA <#ð ~
00002F00	74 CD 5A 6C 69 B5 83 6B 27 EF 5A F4 44 AF 2B 4C	tÍZlipfk'iZðL+L
00002F10	F0 01 F0 01 F0 01 FF 68 D6 46 F0 01 F0 01 2E 61	ð ð ð ÿhÖFð ð .a

我的思路是，把F001出现过的位置中所有十六进制的值单独截取出来，每四位分隔开，把F001替换成0，其他值替换成空格。最后变成下图的8\*125的“图”，其实如果会用CyberChef会更方便，不过我不太习惯。





```

00A0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00B0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00C0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00D0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00E0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00F0h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0100h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0110h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0120h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0130h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0140h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0150h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0160h: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF

```

模板结果 - BMP.bt

名称	值	开始	大小	颜色
> struct BITMAPFILEHEADER bmfh		0h	Eh	Fg: Bg:
▼ struct BITMAPINFOHEADER bmih		Eh	28h	Fg: Bg:
DWORD biSize	40	Eh	4h	Fg: Bg:
LONG biWidth	900	12h	4h	Fg: Bg:
LONG biHeight	250	16h	4h	Fg: Bg:
WORD biPlanes	1	1Ah	2h	Fg: Bg:
WORD biBitCount	24	1Ch	2h	Fg: Bg:
DWORD biCompression	0	1Eh	4h	Fg: Bg:
DWORD biSizeImage	675002	22h	4h	Fg: Bg:
LONG biXPelsPerMeter	2834	26h	4h	Fg: Bg:
LONG biYPelsPerMeter	2834	2Ah	4h	Fg: Bg:
DWORD biClrUsed	0	2Eh	4h	Fg: Bg:
DWORD biClrImportant	0	32h	4h	Fg: Bg:
> struct BITMAPLINE lines[150]		36h	62008h	Fg: Bg:

[https://blog.csdn.net/werkin\\_45696568](https://blog.csdn.net/werkin_45696568)

然后就能看到flag

### misc25

提示: flag在图片下面

png改图片高度还是很简单的, 左框是图片宽度, 右框是图片高度

把00000096改成00000196,看到下面藏的数据

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI	ASCII
00000000	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	PNG	IHDR
00000010	00	00	03	84	00	00	00	96	08	06	00	00	00	EC	9C	CB	"	iacE
00000020	C6	00	00	39	B5	49	44	41	54	78	DA	ED	DD	C1	B1	EA	# 9µIDATxÚiYÁ±ê	
00000030	4C	62	36	60	A5	40	0A	2C	9C	00	19	B8	48	81	2A	47	Lb6`¥@ ,æ ,H *G	

### misc26

提示: flag还是在图片下面, 但到底有多下面?

仍然是png修改高度, 跟上题比数字可以夸张一点

ctfshow{94aef1  
+ True height(hex) of this picture +  
087a7ccf2e28e742efd704c}

flag中间有一段数据是需要计算的, 上网找脚本, 根据图片的crc32值爆破宽高, 这题算出来高度是 25e

### misc27

提示: flag在图片下面

根据提示, 修改图片高度, 看到flag

{there\_is\_no\_flag\_here}

ctfshow{5cc4f19eb01705b99bf41492430a1a14}

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

## misc28

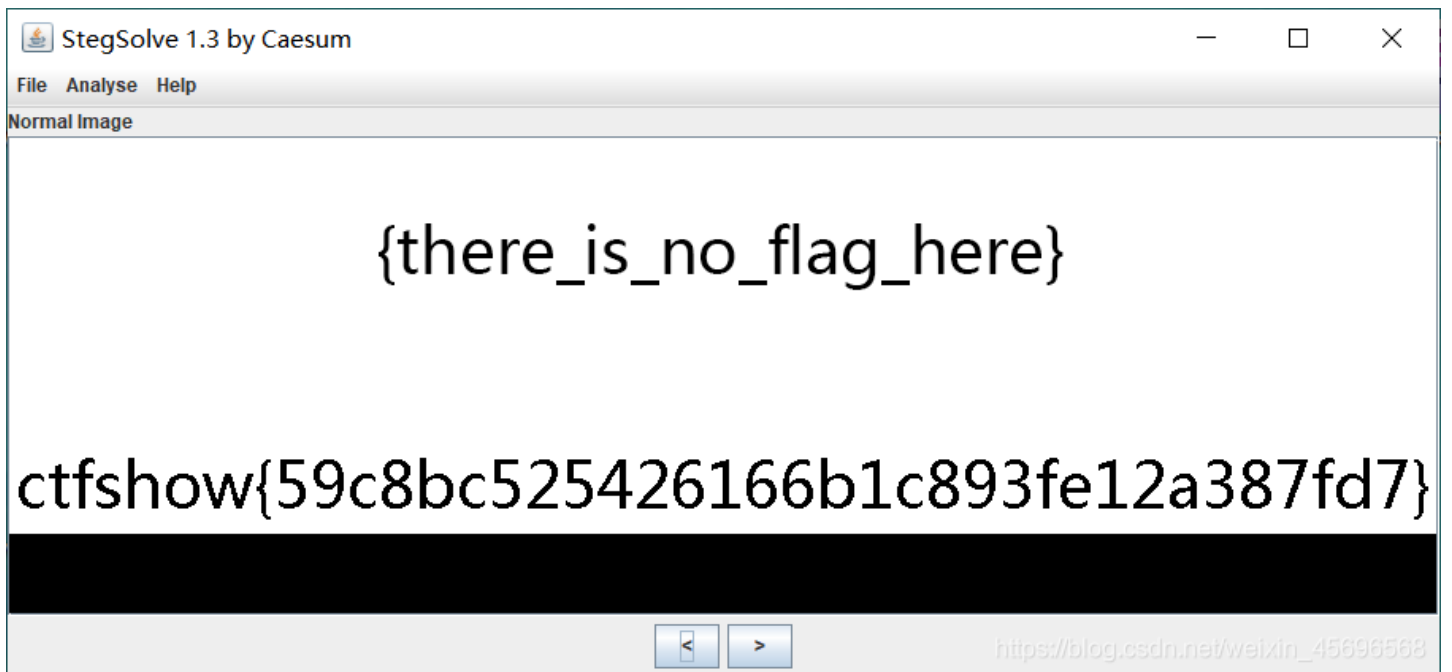
提示: flag在图片下面

## 使用gif.bt修改图片高度

模板结果 - GIF.bt					
名称	值	开始	大小	颜色	
> struct GIFHEADER GifHeader		0h	6h	Fg:	Bg:
> struct LOGICALSCREENDESCRI... ushort Width	900	6h	2h	Fg:	Bg:
ushort Height	150	8h	2h	Fg:	Bg:
> struct LOGICALSCREENDESCRI... UBYTE BackgroundColorIndex	0	Ah	1h	Fg:	Bg:
UBYTE PixelAspectRatio	0	Bh	1h	Fg:	Bg:
> struct GLOBALCOLORTABLE Glo... Dh		Ch	1h	Fg:	Bg:
> struct DATA Data		Dh	60h	Fg:	Bg:
> struct GRAPHICCONTROLEXTE... 6Dh		6Dh	14C8h	Fg:	Bg:
> struct IMAGEDESCRIPTOR Im... 75h		6Dh	8h	Fg:	Bg:
UBYTE ImageSeperator	44	75h	Ah	Fg:	Bg:
ushort ImageLeftPosition	0	75h	1h	Fg:	Bg:
ushort ImageTopPosition	0	76h	2h	Fg:	Bg:
ushort ImageWidth	900	78h	2h	Fg:	Bg:
ushort ImageHeight	300	7Ah	2h	Fg:	Bg:
> struct IMAGEDESCRIPTOR_... 7Eh		7Ch	2h	Fg:	Bg:
> struct IMAGEDATA ImageData		7Eh	1h	Fg:	Bg:
> struct TRAILER Trailer		7Fh	14E6h	Fg:	Bg:
		1535h	1h	Fg:	Bg:

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

改完从预览图中能看到flag，但是直接打开就看不到了，使用图片编辑器或者Stegsolve打开



## misc29

提示: flag在图片下面



我把每一帧的高度都改了，最后发现在第八帧下藏了flag

ushort ImageHeight	900	44E1h	2h	Fg:
> struct IMAGEDESCRIPTOR_...		44E3h	1h	Fg:
> struct LOCALCOLORTABLE Lo...		44E4h	300h	Fg:
> struct IMAGEDATA ImageDat...		47E4h	FA6h	Fg:
> struct GRAPHICCONTROLEXTE...		578Ah	8h	Fg:
▼ struct IMAGEDESCRIPTOR Im...		5792h	Ah	Fg:
UBYTE ImageSeperator	44	5792h	1h	Fg:
ushort ImageLeftPosition	0	5793h	2h	Fg:
ushort ImageTopPosition	0	5795h	2h	Fg:
ushort ImageWidth	900	5797h	2h	Fg:
ushort ImageHeight	900	5799h	2h	Fg:
> struct IMAGEDESCRIPTOR_...		579Bh	1h	Fg:
> struct LOCALCOLORTABLE Lo...		579Ch	300h	Fg:
> struct IMAGEDATA ImageDat...		5A9Ch	6CBh	Fg:
> struct GRAPHICCONTROLEXTE...		6167h	8h	Fg:
▼ struct IMAGEDESCRIPTOR Im...		616Fh	Ah	Fg:
UBYTE ImageSeperator	44	616Fh	1h	Fg:
ushort ImageLeftPosition	0	6170h	2h	Fg:
ushort ImageTopPosition	0	6172h	2h	Fg:
ushort ImageWidth	900	6174h	2h	Fg:
ushort ImageHeight	900	6176h	2h	Fg:
> struct IMAGEDESCRIPTOR_...		6178h	1h	Fg:
> struct LOCALCOLORTABLE Lo...		6179h	300h	Fg:
> struct IMAGEDATA ImageDat...		6479h	6C6h	Fg:

[https://blog.csdn.net/weixin\\_45696588](https://blog.csdn.net/weixin_45696588)



## misc30

**提示：** 正确的宽度是950。

修改宽高即可

名称	值	偏移	大小	颜色
> struct BITMAPFILEHEADER bmfh		0h	Eh	Fg: Bg:
▼ struct BITMAPINFOHEADER bmih		Eh	28h	Fg: Bg:
DWORD biSize	40	Eh	4h	Fg: Bg:
LONG biWidth	950	12h	4h	Fg: Bg:
LONG biHeight	120	16h	4h	Fg: Bg:
WORD biPlanes	1	1Ah	2h	Fg: Bg:
WORD biBitCount	24	1Ch	2h	Fg: Bg:
DWORD biCompression	0	1Eh	4h	Fg: Bg:
DWORD biSizeImage	427802	22h	4h	Fg: Bg:
LONG biXPelsPerMeter	2834	26h	4h	Fg: Bg:

## misc31

提示：高度是正确的，但正确的宽度是多少呢。

思路和misc24一致，最后算出宽度为1082(余数舍去)

## misc32

提示：高度是正确的，但正确的宽度是多少呢

使用以前找脚本

```
import zlib
import struct

# 同时爆破宽度和高度
filename = "misc32.png"
with open(filename, 'rb') as f:
    all_b = f.read()
    data = bytearray(all_b[12:29])
    n = 4095
    for w in range(n):
        width = bytearray(struct.pack('>i', w))
        for h in range(n):
            height = bytearray(struct.pack('>i', h))
            for x in range(4):
                data[x+4] = width[x]
                data[x+8] = height[x]
            crc32result = zlib.crc32(data)
            # 替换成图片的crc
            if crc32result == 0xE14A4C0B:
                print("宽为: ", end = '')
                print(width, end = ' ')
                print(int.from_bytes(width, byteorder='big'))
                print("高为: ", end = '')
                print(height, end = ' ')
                print(int.from_bytes(height, byteorder='big'))
```

```
-----
宽为: bytearray(b'\x00\x00\x04\x14') 1044
高为: bytearray(b'\x00\x00\x00\x96') 150
```

把宽度修改为1044即可看到flag

## misc33

提示：出题人丧心病狂，把高度也改了

还是用上题的脚本

```
-----  
宽为: bytearray(b'\x00\x00\x03\xd2') 978  
高为: bytearray(b'\x00\x00\x00\x8e') 142
```

## misc34

**提示:** 出题人狗急跳墙, 把IHDR块的CRC也改了, 但我们知道正确宽度肯定大于900

把上面的脚本稍微改一下

```
import zlib  
import struct  
filename = "misc34.png"  
with open(filename, 'rb') as f:  
    all_b = f.read()  
    #w = all_b[16:20]  
    #h = all_b[20:24]  
    for i in range(901,1200):  
        name = str(i) + ".png"  
        f1 = open(name, "wb")  
        im = all_b[:16]+struct.pack('>i',i)+all_b[20:]  
        f1.write(im)  
        f1.close()
```

把生成的所有图片都保存下来了(建议在空文件夹里), 然后用眼看哪个是正常的。

最后得到正确的宽度是1123

## misc35

**提示:** 出题人负隅顽抗, 但我们知道正确宽度肯定大于900

总体思路同上题, 不过这题有点小坑, 第一次爆破出的结果没看到flag, 扩大范围之后又跑了一次还是没有, 后面才知道, 要把图片基础的高度调高一点, 才能看到flag

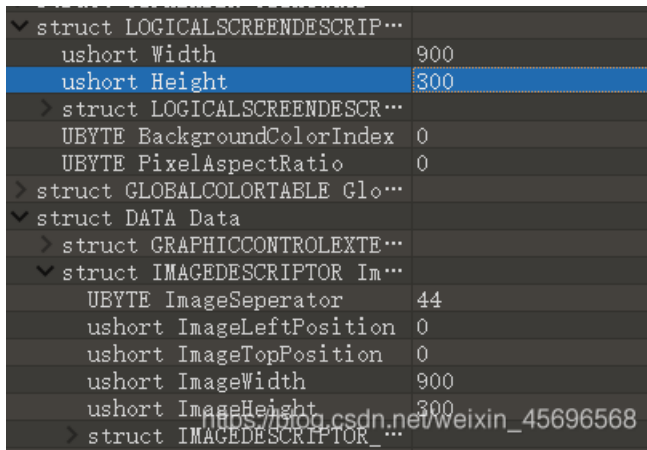
```
import zlib  
import struct  
filename = "misc35.jpg"  
with open(filename, 'rb') as f:  
    all_b = f.read()  
    #w = all_b[159:161]  
    #h = all_b[157:159]  
    for i in range(901,1200):  
        name = str(i) + ".jpg"  
        f1 = open(name, "wb")  
        im = all_b[:159]+struct.pack('>h',i)+all_b[161:]  
        f1.write(im)  
        f1.close()
```

高度我调到了600, 宽度在993-1000这个范围内都可以得到flag

## misc36

**提示:** 出题人坦白从宽, 正确的宽度在920-950之间

吸取上题教训，先把高度调高一点



▼ struct LOGICALSCREENDESCRIP...	
ushort Width	900
ushort Height	300
> struct LOGICALSCREENDESCR...	
UBYTE BackgroundColorIndex	0
UBYTE PixelAspectRatio	0
> struct GLOBALCOLORTABLE Glo...	
▼ struct DATA Data	
> struct GRAPHICCONTROLEXTE...	
▼ struct IMAGEDESCRIPTOR Im...	
UBYTE ImageSeperator	44
ushort ImageLeftPosition	0
ushort ImageTopPosition	0
ushort ImageWidth	900
ushort ImageHeight	300
> struct IMAGEDESCRIPTOR_...	

脚本如下：

```
import zlib
import struct
filename = "misc36.gif"
with open(filename, 'rb') as f:
    all_b = f.read()
    for i in range(920,951):
        name = str(i) + ".gif"
        f1 = open(name,"wb")
        im = all_b[:38]+struct.pack('>h',i)[::-1]+all_b[40:]
        f1.write(im)
        f1.close()
```

正确宽度是941

## misc37

**提示：** flag在图片里

gif文件，用 [Stegsolve](#) 打开，一共有45帧，flag藏在 9、14、21、31、34 帧里

## misc38

**提示：** flag在图片里

这个是apng图片，像gif一样会动的，用浏览器打开就知道了

可以使用 [APNG Disassembler](#) 来把每一帧分离出来，9、17、36、40 帧中藏有flag

## misc39

这题也是一个gif，不过这里是利用 **不同帧之间的间隔时间** 来隐写的。

这里利用linux下的工具 [identify](#)

安装命令：sudo apt-get install imagemagick

基本的命令格式：

```
identify [options] input-file
identify:命令名称
options:参数
input-file:文件名。
```

提取命令：identify -format "%T " misc39.gif > 1.txt



用010editor打开，发现有48个IDAT块，结合提示flag长度为41，很有可能有关联。  
用tweakpng打开misc42.png，发现这七个数正好是ctfshow对应的ascii码

File	Edit	Insert	Options	Tools	Help
Chunk	Length	CRC	Attributes	Contents	
IHDR	13	09dad...	critical	PNG image header: 900×150, 8 bits/sample,	
IDAT	229	82402...	critical	PNG image data	
IDAT	152	b92a4...	critical	PNG image data	
IDAT	191	c2947...	critical	PNG image data	
IDAT	229	edf7ec...	critical	PNG image data	
IDAT	152	27413...	critical	PNG image data	
IDAT	191	e625b...	critical	PNG image data	
IDAT	49	19eb9...	critical	PNG image data	
IDAT	99	d639e...	critical	PNG image data	
IDAT	116	af63a2...	critical	PNG image data	
IDAT	102	d7127...	critical	PNG image data	
IDAT	115	b5296...	critical	PNG image data	
IDAT	104	dce9d...	critical	PNG image data	
IDAT	111	302ca...	critical	PNG image data	
IDAT	119	927d6...	critical	PNG image data	
IDAT	123	6ef517...	critical	PNG image data	
IDAT	48	98574...	critical	PNG image data	
IDAT	55	866b9...	critical	PNG image data	
IDAT	56	b7453...	critical	PNG image data	
IDAT	99	4fb61...	critical	PNG image data	
IDAT	98	5a119f...	critical	PNG image data	
IDAT	100	657dd...	critical	PNG image data	
IDAT	48	285d6...	critical	PNG image data	
IDAT	102	004bb...	critical	PNG image data	

把剩下的数也转换，得到flag

## misc43

**提示：** 错误中隐藏着通往正确答案的道路

用tweakpng打开，报了一堆错，然后使用pngdebugger分析，发现所有IDAT块 crc32 值都是错误的

```
C:\Windows\System32\cmd.exe

0x00000021      chunk-length=0x00000180 (384)
0x00000025      chunk-type=' IDAT'
0x000001A9      crc-code=0xE59387E5
>> (CRC CHECK)  crc-computed=0x8385F691      =>      CRC FAILED

0x000001AD      chunk-length=0x00000180 (384)
0x000001B1      chunk-type=' IDAT'
0x00000335      crc-code=0x93A62E63
>> (CRC CHECK)  crc-computed=0x42434298      =>      CRC FAILED

0x00000339      chunk-length=0x00000180 (384)
0x0000033D      chunk-type=' IDAT'
0x000004C1      crc-code=0x74667368
>> (CRC CHECK)  crc-computed=0x4462C3A1      =>      CRC FAILED

0x000004C5      chunk-length=0x00000180 (384)
0x000004C9      chunk-type=' IDAT'
0x0000064D      crc-code=0x6F777B36
>> (CRC CHECK)  crc-computed=0x397611E1      =>      CRC FAILED

0x00000651      chunk-length=0x00000180 (384)
0x00000655      chunk-type=' IDAT'
0x000007D9      crc-code=0x65623235
>> (CRC CHECK)  crc-computed=0x4F02AFA2      =>      CRC FAILED

https://blog.csdn.net/weixin_45696568
```

结合提示，把报错的crc提取出来，hex转字符得到flag

```
Input
lines: 1

E59387E593A62E63746673686F777B36656232353839666666666356533393066653662383735303464626330

Output
time: 0ms
length: 44
lines: 1

哇哦.ctfshow{6eb2589ffff5e390fe6b87504dbc0892}
```



提示：错误中还隐藏着坑

根据提示，可以知道还是从crc32入手，先用 `010editor` 打开，好家伙344个IDAT块

> struct PNG_CHUNK chunk[323]	IDAT (Critical, Public, Unsafe to Copy)	5B839h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[324]	IDAT (Critical, Public, Unsafe to Copy)	5BCC5h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[325]	IDAT (Critical, Public, Unsafe to Copy)	5C151h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[326]	IDAT (Critical, Public, Unsafe to Copy)	5C5DDh	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[327]	IDAT (Critical, Public, Unsafe to Copy)	5CA69h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[328]	IDAT (Critical, Public, Unsafe to Copy)	5CF55h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[329]	IDAT (Critical, Public, Unsafe to Copy)	5D381h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[330]	IDAT (Critical, Public, Unsafe to Copy)	5D80Dh	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[331]	IDAT (Critical, Public, Unsafe to Copy)	5DC99h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[332]	IDAT (Critical, Public, Unsafe to Copy)	5E125h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[333]	IDAT (Critical, Public, Unsafe to Copy)	5E5E1h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[334]	IDAT (Critical, Public, Unsafe to Copy)	5EA3Dh	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[335]	IDAT (Critical, Public, Unsafe to Copy)	5EEC9h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[336]	IDAT (Critical, Public, Unsafe to Copy)	5F355h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[337]	IDAT (Critical, Public, Unsafe to Copy)	5F7E1h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[338]	IDAT (Critical, Public, Unsafe to Copy)	5FC6Dh	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[339]	IDAT (Critical, Public, Unsafe to Copy)	600F9h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[340]	IDAT (Critical, Public, Unsafe to Copy)	60585h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[341]	IDAT (Critical, Public, Unsafe to Copy)	60A11h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[342]	IDAT (Critical, Public, Unsafe to Copy)	60E9Dh	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[343]	IDAT (Critical, Public, Unsafe to Copy)	61329h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[344]	IDAT (Critical, Public, Unsafe to Copy)	61835h	48Ch	Fg:	Bg:
> struct PNG_CHUNK chunk[345]	IDAT (Critical, Public, Unsafe to Copy)	61D41h	48Ch	Fg:	Bg:

建议别用 `tweakpng` 打开，因为每有一个IDAT块的crc32是错误的，就会弹窗一次，这题少说得有一百来个？

这里使用 `PNGDebugger`，用了一个小技巧

```
PNGDebugger.exe misc44.png > 1.txt
```

然后写脚本，把 CRC OK的替换成1，CRC FAILED替换成0

```
1.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
>> (CRC CHECK) crc-computed=0xC8255A59 ==> CRC OK!

0x00004455      chunk-length=0x00000480 (1152)
0x00004459      chunk-type='IDAT'
0x000048DD      crc-code=0x238597F3
>> (CRC CHECK) crc-computed=0x238597F3 ==> CRC OK!

0x000048E1      chunk-length=0x00000480 (1152)
0x000048E5      chunk-type='IDAT'
0x00004D69      crc-code=0x1255BA67
>> (CRC CHECK) crc-computed=0x25BE9607 ==> CRC FAILED

0x00004D6D      chunk-length=0x00000480 (1152)
0x00004D71      chunk-type='IDAT'
0x000051F5      crc-code=0xF8D4A203
>> (CRC CHECK) crc-computed=0xF8D4A203 ==> CRC OK!

第 1 行, 第 1 列    100%    Windows (CRLF)    UTF-8
```

使用脚本前，先把前十行无用的内容删去，再把最后四行也删去。

```
f=open("1.txt","r")
s=f.read()
f.close()
flag=""
for i in s.split():
    if "OK!" == i:
        flag += "1"
    elif "FAILED" ==i:
        flag += "0"
print(flag)
#1111111111111111111101100011011101000110011001110011011010000110111101110111101110110001101100011001100010110000
1011001100011001100110011000100110011000111001001101100011001100110000001110000110011001100011001100010011001
0001101100011001100110010001100110011000101100010011001010011011100111000001100110110011000110110001110010110010
101111101
print(len(flag)) #344
for i in range(43):
    print(chr(int(flag[8*i:8*(i+1)],2)),end="")
```

## misc45

**提示：** 有时候也需要换一换思维格式

其他题都出来了，这个题迟迟没有被解出来，我本来以为是脑洞题，十天前被bit师傅拿了一血，原来是一个新的知识点。

八神很贴心地告诉我们要 **换图片的格式**，具体做法就是：先去在线网站把图片从 **png** 格式转为 **bmp** 格式，然后直接binwalk提取就能得到flag.png了。

```
(volcano@kali)-[~]
└─$ binwalk /home/volcano/桌面/misc45.bmp
```

DECIMAL	HEXADECIMAL	DESCRIPTION
65620	0x10054	gzip compressed data, has original file name: "flag.png", from Unix, last modified: 2021-03-29 15:44:52

bmp格式下，中间的位置插入了一个gzip的数据，直接肉眼看很难看出来，至于为什么原本的png格式下，binwalk得不到结果呢？大师傅们的解释是png和bmp **像素点的读取方式不一样**。

```
(volcano@kali)-[~]
└─$ binwalk /home/volcano/桌面/misc45.png
```

DECIMAL	HEXADECIMAL	DESCRIPTION
0	0x0	PNG image, 900 x 150, 8-bit/color RGB, non-interlaced

## misc46

是一个gif

提取出它的详细信息：`identify misc46.gif > /1.txt` (这里直接在根目录生成1.txt，好找)

内容大概长这样，其中 `0+0`、`174+49`、`196+47` 这些是**偏移量**，这题就利用这个来作图。

— □

) 查看(V) 帮助(H)

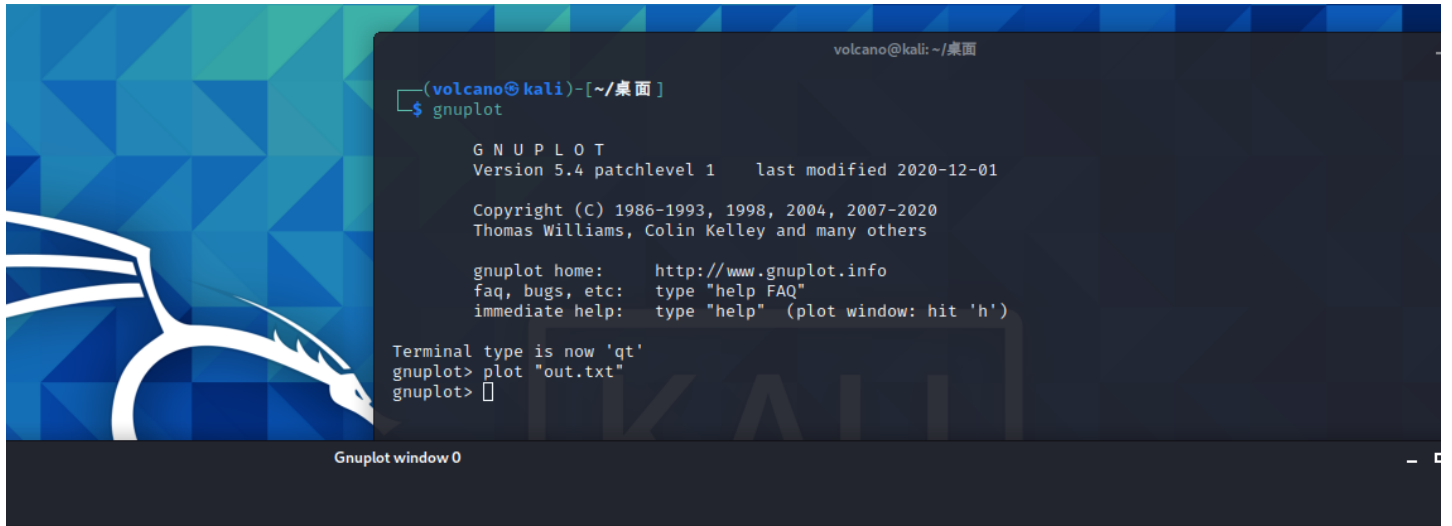
```
面/misc46.gif[0] GIF 900x150 900x150+0+0 8-bit sRGB 2c 0.040u 0:00.050
面/misc46.gif[1] GIF 450x50 900x150+174+49 8-bit sRGB 16c 0.040u 0:00.054
面/misc46.gif[2] GIF 450x50 900x150+196+47 8-bit sRGB 16c 0.040u 0:00.054
面/misc46.gif[3] GIF 450x50 900x150+256+49 8-bit sRGB 16c 0.040u 0:00.054
面/misc46.gif[4] GIF 450x50 900x150+293+52 8-bit sRGB 16c 0.040u 0:00.054
面/misc46.gif[5] GIF 450x50 900x150+220+49 8-bit sRGB 16c 0.040u 0:00.054
```

写一个很简单的脚本把坐标提取出来

```
f = open("1.txt", "r")
x = f.readlines()
f.close()

f = open("out.txt", "w")
for i in x:
    f.write(i.split("+")[1])
    f.write(" ")
    f.write(i.split("+")[2][:2])
    f.write("\n")
f.close()
```

再利用 `gnuplot` 作图，画出来的结果有点模糊，自行调整



再翻转一下得到flag

## misc47

给了一个png，打开发现没内容，用浏览器打开，确认是apng

思路和上题一致，不过稍微复杂一点，先通过这篇文章了解一下apng文件结构，简单来说就是每一个 `IDAT` 块前面都会有一个 `fcTL` 块，它其中就包含水平垂直偏移量

如下，对应坐标点就是(182,52)

> struct PNG_CHUNK chunk[1]	acTL (Ancillary, Private, Unsafe to Copy)	21h	14h	Fg:	Bg:
> struct PNG_CHUNK chunk[2]	fcTL (Ancillary, Private, Unsafe to Copy)	35h	26h	Fg:	Bg:
> struct PNG_CHUNK chunk[3]	IDAT (Critical, Public, Unsafe to Copy)	5Bh	2F1h	Fg:	Bg:
▼ struct PNG_CHUNK chunk[4]	fcTL (Ancillary, Private, Unsafe to Copy)	34Ch	26h	Fg:	Bg:
uint32 length	26	34Ch	4h	Fg:	Bg:
> union CTYPE type	fcTL	350h	4h	Fg:	Bg:
▼ struct PNG_CHUNK_FCTL fctl		354h	1Ah	Fg:	Bg:
uint32 sequence_number	1	354h	4h	Fg:	Bg:
uint32 width	450	358h	4h	Fg:	Bg:
uint32 height	50	35Ch	4h	Fg:	Bg:
uint32 x_offset	182	360h	4h	Fg:	Bg:
uint32 y_offset	52	364h	4h	Fg:	Bg:
int16 delay_num	100	368h	2h	Fg:	Bg:
int16 delay_den	1000	36Ah	2h	Fg:	Bg:
enum APNG_DISPOSE_OP di...	APNG_DISPOSE_OP_NONE (0)	36Ch	1h	Fg:	Bg:
enum APNG_BLEND_OP blen...	APNG_BLEND_OP_SOURCE (0)	36Dh	1h	Fg:	Bg:
uint32 crc	EDD8A7h	36Fh	4h	Fg:	Bg:

之前的脚本有的师傅使用的时候有点小问题，所以改了一下

```

from PIL import Image

im = Image.new('RGB', (400, 80), 255)
f = open('1.txt', 'r') #把图片的十六进制导出, 保存为1.txt
c = f.read()
f.close()

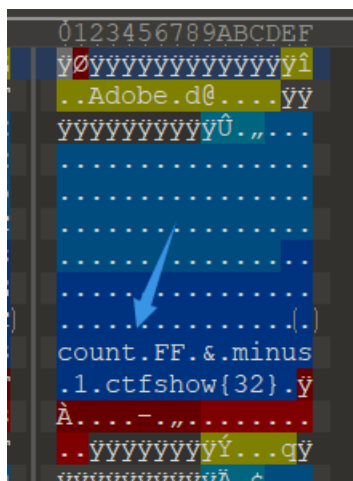
lt = c.split("6663544C")
for i in range(2, len(lt)):
    x = int(lt[i][28:32], 16)
    y = int(lt[i][36:40], 16)
    im.putpixel((x, y), 0)
im.show()

```

## misc48

用010editor打开, 发现有提示

- 1、统计FF的数量, 再减去1
- 2、ctfshow{}中包含32个字符



提示了, 但没有完全提示, 因为第一条提示, 其实指的是 **统计每两个有意义块之间的FF的数量再减一**

图中紫色的就是，开头的那个FF也算，因为只有一个，减去1后就是0；接下来是12、11、0...



因为flag长度是32位，所以只统计前32个，即：

```
0 12 11 0 7 10 13 13 9 0 9 13 0 13 6 0 10 9 2 1 0 1 10 8 11 5 12 7 2 2 3 10
```

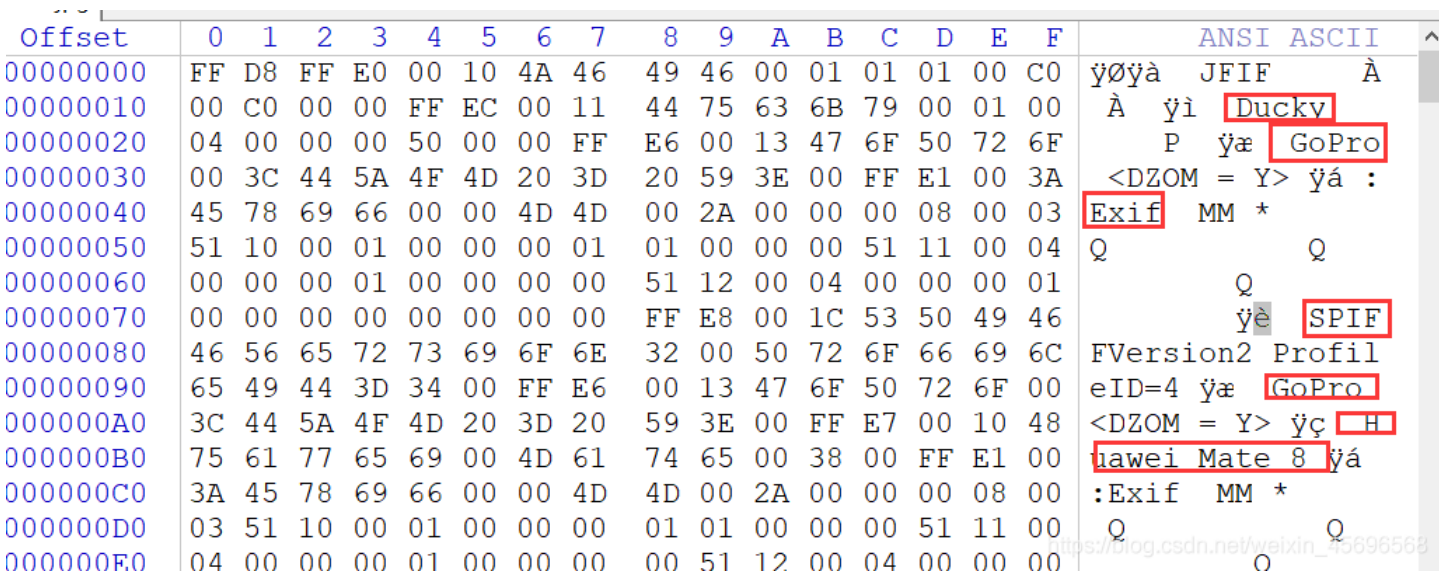
分别转十六进制后，再连接在一起，得到：`ctfshow{0cb07add909d0d60a92101a8b5c7223a}`

## misc49

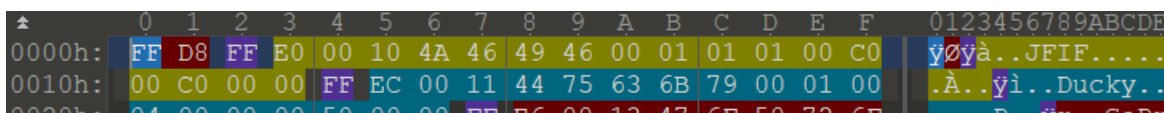
提示：它们一来就是十六种。本题略脑洞，可跳过

八神的脑洞题，靠我自己想是不行的，果断参考wp

用winhex打开，能看到很多字符串，这其实是八神给的提示，虽然我没get到



重点是这些字符串前面，都出现过FFE? 这种格式的数据，搜索一下发现有挺多的



0020h:	04 00 00 00 30 00 00 FF	E8 00 13 47 0F 50 72 0F	....F..ÿæ..GOF
0030h:	00 3C 44 5A 4F 4D 20 3D	20 59 3E 00 FF E1 00 3A	.<DZOM = Y>.ÿá.
0040h:	45 78 69 66 00 00 4D 4D	00 2A 00 00 00 08 00 03	Exif..MM.*.....
0050h:	51 10 00 01 00 00 00 01	01 00 00 00 51 11 00 04	Q.....Q..
0060h:	00 00 00 01 00 00 00 00	51 12 00 04 00 00 00 01	.....Q.....
0070h:	00 00 00 00 00 00 00 00	FF E8 00 1C 53 50 49 46	(....)....ÿè..SPI
0080h:	46 56 65 72 73 69 6F 6E	32 00 50 72 6F 66 69 6C	FVersion2.Profi
0090h:	65 49 44 3D 34 00 FF E6	00 13 47 6F 50 72 6F 00	eID=4.ÿæ..GoPro
00A0h:	3C 44 5A 4F 4D 20 3D 20	59 3E 00 FF E7 00 10 48	<DZOM = Y>.ÿç.
00B0h:	75 61 77 65 69 00 4D 61	74 65 00 38 00 FF E1 00	uawei.Mate.8.ÿá
00C0h:	3A 45 78 69 66 00 00 4D	4D 00 2A 00 00 00 08 00	:Exif..MM.*.....
00D0h:	03 51 10 00 01 00 00 00	01 01 00 00 00 51 11 00	.Q.....Q.
00E0h:	04 00 00 00 01 00 00 00	00 51 12 00 04 00 00 00	.....Q.....
00F0h:	01 00 00 00 00 00 00 00	00 FF EA 00 28 50 68 6F	.....ÿê.(Ph
0100h:	74 6F 53 74 75 64 69 6F	E5 A5 97 E5 A8 83 E7 BC	toStudioâ¥-â"fc
0110h:	9D E5 90 88 EF BC 8C E7	8B 97 E9 83 BD E4 B8 8D	.â.^i¿Eç<-éf¿â.
0120h:	E5 81 9A FF E1 00 3A 45	78 69 66 00 00 4D 4D 00	â.šÿá.:Exif..MM
0130h:	2A 00 00 00 08 00 03 51	10 00 01 00 00 00 01 01	*.....Q.....
0140h:	00 00 00 51 11 00 04 00	00 00 01 00 00 00 00 51	...Q.....
0150h:	12 00 04 00 00 00 01 00	00 00 00 00 00 00 00 FF	.....
0160h:	E5 00 11 53 75 6D 73 75	6E 67 00 42 6F 6D 62 00	â..Samsung.Bomb
0170h:	37 00 FF E3 00 13 4D 45	54 41 00 00 4B 6F 64 61	7.ÿã..META..Kod
0180h:	6B 00 54 2D 30 33 00 FF	EF 00 10 47 72 61 70 68	k.T-03.ÿi..Grap
0190h:	43 6F 6E 76 00 EA 02 71	00 FF E5 00 11 53 75 6D	Conv.ê.q.ÿã..Su
01A0h:	73 75 6E 67 00 42 6F 6D	62 00 37 00 FF ED 00 38	sung.Bomb.7.ÿí.
01B0h:	50 68 6F 74 6F 73 68 6F	70 20 33 2E 30 00 38 42	Photoshop 3.0.8
01C0h:	49 4D 04 04 00 00 00 00	00 00 38 42 49 4D 04 25	IM.....8BIM.
01D0h:	00 00 00 00 00 10 D4 1D	8C D9 8F 00 B2 04 E9 80	.....Ô.ËÛ..².é
01E0h:	09 98 EC F8 42 7E FF EA	00 28 50 68 6F 74 6F 53	.~iøB~ÿê.(Photo
01F0h:	74 75 64 69 6F E5 A5 97	E5 A8 83 E7 BC 9D E5 90	tudioâ¥-â"fc¿.â
0200h:	88 EF BC 8C E7 8B 97 E9	83 BD E4 B8 8D E5 81 9A	^i¿Eç<-éf¿â..â.
0210h:	FF E3 00 13 4D 45 54 41	00 00 4B 6F 64 61 6B 00	ÿã..META..Kodak
0220h:	54 2D 30 33 00 FF E9 00	15 4D 65 64 69 61 4A 75	T-03.ÿé..MediaJ
0230h:	6B 65 62 6F 78 00 41 6C	62 75 6D 00 FF E4 00 11	kebox.Album.ÿã.
0240h:	46 6C 61 73 68 50 69 78	00 31 2E 30 2E 30 00 FF	FlashPix.1.0.0.
0250h:	E8 00 1C 53 50 49 46 46	56 65 72 73 69 6F 6E 32	è..SPIFFVersion
0260h:	00 50 72 6F 66 69 6C 65	49 44 3D 34 00 FF EF 00	.ProfileID=4.ÿi

查找结果

地址	值
已找到 289 个 'FFE'.	
0h	FFE
01	FFE

[https://blog.csdn.net/weixin\\_45696568](https://blog.csdn.net/weixin_45696568)

把所有十六进制数保存在2.txt中，用一个小脚本处理一下

```
f=open("2.txt","r")
txt=f.read().replace("\n","")
f.close()

l=txt.split("FFE")
flag=""
for i in range(1,len(l)):
    flag += l[i][0]
print(flag.lower()[:32]) #得到的结果套上ctfshow{}
```

其实就是把FFE后面的那个字符提取出来，再连接在一起，一共32位(), 这就是flag。

这里写脚本的时候有个小失误，把这种也统计进去了，所以只有前32位是符合格式的正确。

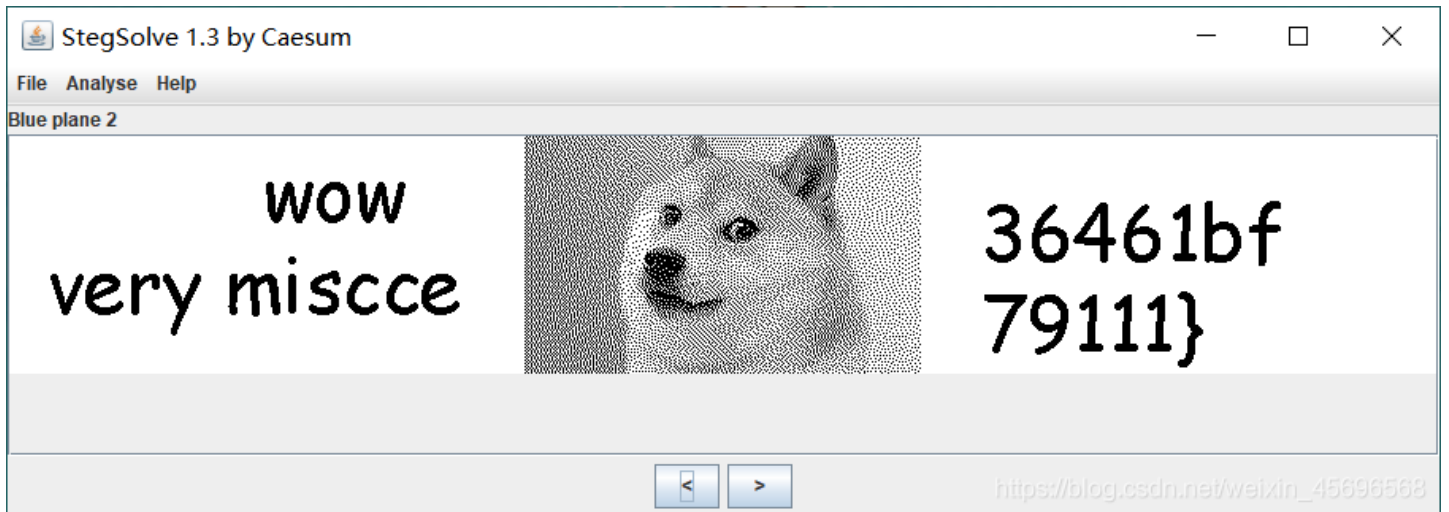
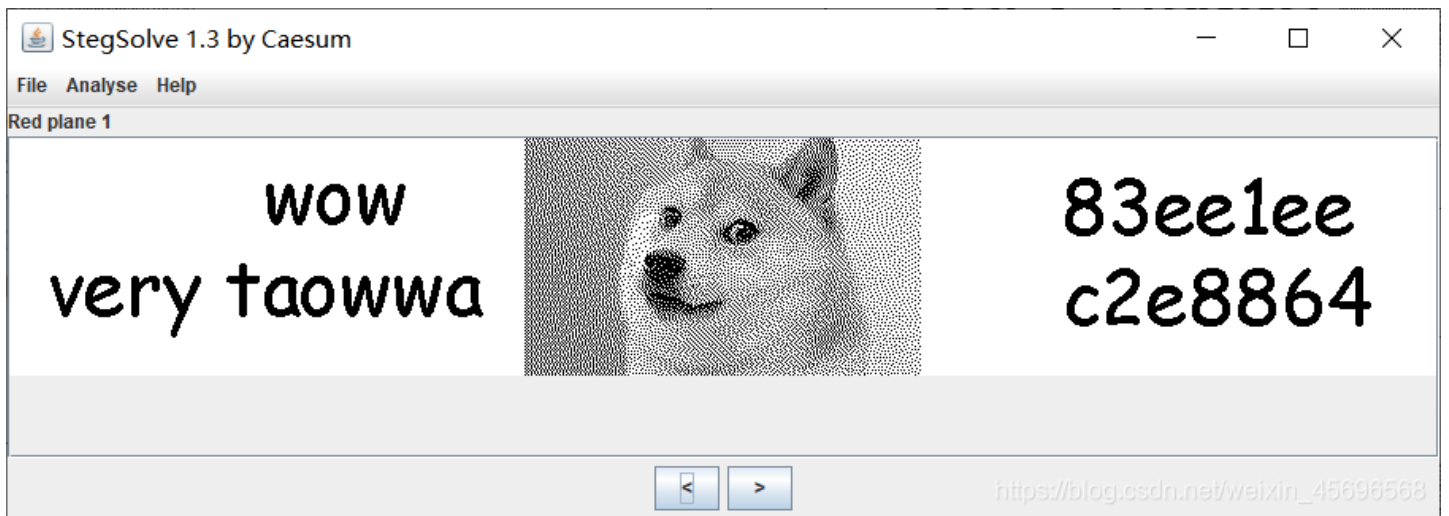
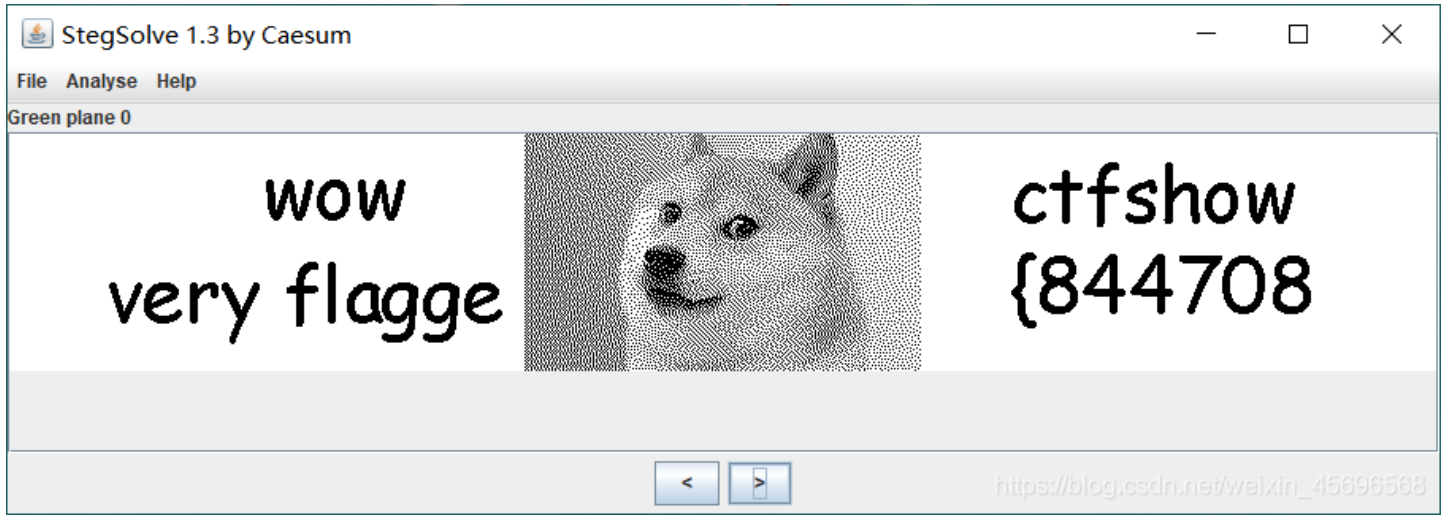
	C	D	E	F	
94	92	DE	E5	¥N.	
E0	E7	FC	E3	b-'	
CF	CC	3F	CC	±œÿ	
9A	6E	9D	E5	l~	
1B	9B	EB	C9	o*M	
FC	A4	12	10	@a,	
FB	5F	FE	7D	c+V	
36	E6	E7	CB	×Ëp	
60	37	86	10	"÷	

## 图片篇(颜色通道)

### misc50



考察 Stegsolve 的使用



flag: `ctfshow{84470883ee1eec2e886436461bf79111}`